LiveSim: Going Live with Microarchitecture Simulation

Sina Hassani, Gabriel Southern and Jose Renau
Dept. of Computer Engineering, University of California, Santa Cruz
{sihassan,gsouther,renau}@ucsc.edu

ABSTRACT

Computer architects rely heavily on software-based microarchitecture simulators, which typically take hours or days to produce results. We have developed LiveSim, a novel microarchitectural simulation methodology that provides simulation results within seconds, making it suitable for interactive use.

LiveSim works by creating in-memory checkpoints of application state, and then executing randomly selected samples from these checkpoints in parallel to produce simulation results. The initial results, which we call LiveSample, are reported less than one second after starting the simulation. As more samples are simulated the results become more accurate and are updated in real-time. Once enough samples are gathered, LiveSim provides confidence intervals for the reported values and continues simulation until it reaches the target confidence level, which we call LiveCI.

We evaluated LiveSim using SPEC CPU 2006 benchmarks and found that within 5 seconds after starting simulation, LiveSample results reached an average error of 3.51% compared to full simulation, and the LiveCI results were available within 41 seconds on average.

1. INTRODUCTION

Computer architects are constrained by the fact that system design is a slow, expensive, and time-consuming process. To ameliorate this architects use a variety of techniques to prototype ideas and perform design space exploration. One of the most important techniques is architectural simulation where a software model of the simulated system is developed to evaluate its performance using realistic benchmarks. Unfortunately, software simulation is many orders of magnitude slower than the real systems being designed. This limits the length of the benchmarks that can be executed, and also forces architects to wait for long periods (from minutes to days) until new simulation results are ready.

Many techniques have been developed to reduce simulation time including: benchmarks size reduction [1], specialized hardware [2], phase-based sampling [3], and statistical sampling [4]. Of these techniques, the sampling based approaches typically provide the best trade-off between simulation fidelity, speed, and flexibility.

The state of the art simulation techniques have reduced simulation time from weeks to days or hours, but in many ways microarchitectural simulation still uses the methodology of the era of punched cards and batch queues. An architect typically first develops and configures the simulation parameters. Then the simulation is submitted to a batch queue and runs for hours or days while the architect works on something else. After the simulation finishes execution the architect must recall what she was working on, interpret the results and then repeat the cycle with new experiments.

This methodology contrasts with the rapid development techniques popular in many types of software engineering. We expect the productivity of computer architects to improve with an interactive development environment. To support this development model we propose LiveSim, a simulation framework that provides simulation results in near real-time. In this paper we define our near real-time goal as within 5 seconds and we use the terms live and interactive[1] interchangeably. LiveSim provides initial results as soon as the first sample finishes simulation, and it provides a confidence interval to bound the expected error after a minimum number of samples have executed (which is typically within 5 seconds on our system). If necessary LiveSim continues simulating more samples until reaching a user defined threshold for the confidence interval. We call the results that are updated in real-time LiveSample, and the result that is within the desired confidence interval LiveCI.

LiveSim works by first running a setup phase that executes the applications in emulation only mode and creates in-memory checkpoints with architectural state. This step is completely independent of simulated microarchitecture. Next the microarchitecture simulator is loaded as a dynamic library, which allows it to be easily modified without rerunning the costly setup phase. Then a calibration phase runs which executes a sample from each checkpoint using the current simulator configuration. The samples are used to characterize the checkpoint and allow for clustering. Although the measured performance depends somewhat on the simulated microarchitecture, in practice the clustering tends to be

[1]The simulator is used interactively, not the benchmark that is simulated.

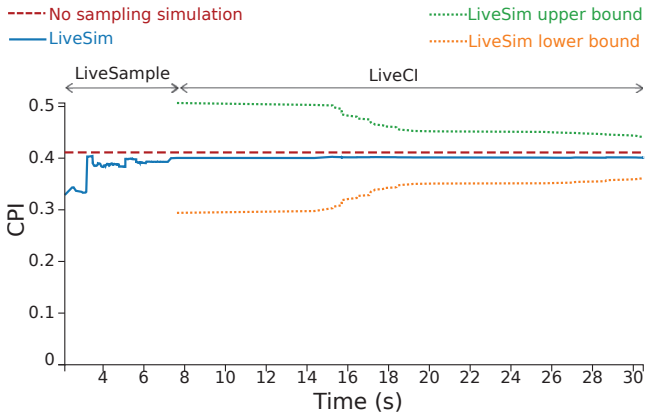


Figure 1: LiveSim timeline showing how the user is presented with LiveSample results from the beginning, how accurate the results get within a few seconds, and how the simulation continues running until the confidence interval reaches the threshold for the LiveCI results.

associated with program phases, and as a result the calibration phase tends not to need to be repeated even if the simulated microarchitecture has radical changes. At this point LiveSim is ready for interactive use allowing the user to experiment with changes to the simulated microarchitecture. After making a change to the simulator the user requests new simulation results. LiveSim randomly selects the minimum number of checkpoints and begins simulating samples from the selected checkpoints in parallel and reporting the LiveSample results to the user. After meeting the cutoff for the LiveCI results the simulation halts and reports the final results.

Figure 1 illustrates an example of how LiveSim works by showing the simulation result of running namd benchmark from SPEC2006 for approximately 10 seconds in a Haswell CPU and simulating a similar CPU with LiveSim. Unlike traditional simulators, LiveSim starts to produce results as each simulation sample finishes (LiveSample). As the evaluation will show, after 5 seconds it is able to provide results that are within 4% of the correct result. The correct result is a full simulation of the 10 seconds without sampling shown as No Sampling Simulation in the figure. Once enough samples are gathered, it is possible to start reporting the confidence interval for the simulation. As more samples are added, the confidence interval decreases and stops the simulation when enough samples are processed. While LiveSample provides accurate results in a very short time, it cannot guarantee them. Live Confidence Interval (LiveCI) bounds the error according to the user requested acceptable error.

In this paper we make the following novel contributions:

- Introduce LiveSim, a new architectural simulation methodology that enables interactive microarchitecture design space exploration.
- Demonstrate that LiveSim is able to provide very accurate LiveSample simulation results within 5 seconds. These results are independent of the length of the simulated benchmark and simulating more instructions does not increase the time it takes for LiveSim to provide the LiveSample results.
- Demonstrate that LiveSim is able to produce LiveCI results that bound the simulation error within 10% within 41 seconds on average.

The rest of this paper is organized as follows: Section 2 provides some background about existing techniques to speed up simulation; Section 3 explains how LiveSim works; Section 4 details the setup of our evaluation framework; Section 5 describes our results; Section 6 provides a comparison with related work; and Section 7 concludes.

2. BACKGROUND

Most architectural simulators are implemented as discrete event simulators where the simulator models the changes to microarchitectural state that occur each clock cycle while the simulated processor executes an instruction. Simulating a single instruction can require the host to execute thousands of instructions to update all of the simulated microarchitectural state for an advanced out-of-order processor, and even fast simulators are thousands of times slower than native execution.

There are a variety of ways to cope with the slow simulation speed. One is to simulate benchmarks that execute very small numbers of instructions; however, it is difficult to ensure that these results are comparable to those obtained with standard benchmark inputs [5, 1]. Another approach is to accelerate the timing simulation using FPGAs [2, 6, 7], but this requires custom hardware, increases simulator development complexity, and is not widely used in practice. The most popular approach is to use sampling to reduce the number of instructions that need to be simulated, and this is the technique we use for LiveSim.

Many simulators have a variety of levels of simulation detail ranging from the most detailed mode which models all microarchitectural details, to modes that only simulate structures with long lived state (such as caches or branch predictor), to emulation-only mode, or even modes that run parts of the simulated benchmark directly on the host system [8]. As the level of detail decreases, the speed of the simulation increases. Most sampling techniques take advantage of this difference in simulation speed by executing the majority of instructions in a faster simulation mode, and extrapolating statistics collected from a small percentage of instructions executed using full detailed simulation mode. The two main sampling techniques are profile based sampling and statistical sampling.

Profile based sampling attempts to identify a few regions of a benchmark that are representative of the behavior of the full benchmark execution. Sherwood et al. [3] developed SimPoint, which works by profiling a benchmark and collecting information about the distribution of basic blocks executed during benchmark

execution. This information is used to find phases in program execution and then select representative samples for each phase. The statistics that are collected from these samples can be used to extrapolate results that tend to be very close to those from full benchmark execution. The effectiveness of the original SimPoint proposal was purely heuristic based, but Perelman et al. [9] extended SimPoint to provide statistical confidence measures.

Wunderlich, et al. [4] developed the SMARTS framework, which applies statistical sampling theory to computer architecture simulation. The main drawback with SMARTS is that it requires continuous updates to the simulated cache and branch predictor microarchitectural state between sampling units. The simulation mode that updates this state is called functional warming, and while it is faster than detailed simulation, it is still much slower than native execution. Although SMARTS is the de facto reference for applying statistical sampling to microarchitecture simulation, earlier work from Conte et al. [10, 11] also explored using statistical sampling with microarchitecture simulation.

Since functional warming of the cache dominates simulation time there have been a variety of proposals to reduce the amount of warmup that is needed and to speed up the emulation mode. One technique is to save some of the simulation state in a checkpoint and then load this checkpoint during future simulation runs [12, 13]. Another technique is to forego detailed cache modeling during the functional warmup phase and simply record the sequence of memory operations. This information can then be used to quickly rebuild the cache state prior to detailed simulation of a sampling unit [14].

LiveSim builds on existing work that uses sampling to accelerate microarchitecture simulation, but rather than simply trying to make the simulation faster, it is designed to be suitable for interactive use. LiveSim uses statistical sampling, in-memory checkpoints, checkpoint clustering, parallel checkpoint execution, and a fast cache warmup technique in order to support interactive simulation.

3. LIVESIM METHODOLOGY

The previous section provided some background about the sampling techniques that LiveSim leverages to enable interactive or Live simulation. In this section we explain in detail exactly how we implemented LiveSim. The basic outline for how LiveSim works is as follows:

- **Setup**: Run simulated benchmark in emulation only mode and periodically create in-memory checkpoints that contain architecturally visible state. Information about the sequence of memory accesses is also recorded during checkpoint creation and used later for cache warmup. The state contained in the checkpoints is independent of the simulated microarchitecture.
- **Calibration**: Execute a sample from each of the checkpoints and use the recorded statistics to group the checkpoints into clusters.
- **LiveSample**: When the user requests new simulation results LiveSim begins executing a number of checkpoints in parallel. As soon as simulation results are ready they are reported visually to the user.
- **LiveCI**: After a minimum number of checkpoints complete execution LiveSim monitors the calculated confidence interval. If it is not within the user specified limit then LiveSim randomly selects more checkpoints to run. Eventually the confidence interval reaches the selected limit and the simulation halts.

In LiveSim *checkpoints* contain all of the architecturally visible state necessary to start simulation from a specific point in the benchmark and can be reused multiple times with many different simulator configurations, whereas *samples* represent the result of simulating a specific microarchitecture for a specific checkpoint. Samples are created by first copying the checkpoint state, next loading the simulator library and configuration for the microarchitecture that is being simulated, next warming up the microarchitecture state, and finally collecting statistics for the sample.

In the rest of this section we explain in detail how each of the steps in LiveSim works.

3.1 Sampling Setup

The field of inferential statistics provides well known techniques for inferring statistics about a population given a sample of that population. SMARTS [4] demonstrated that systematic sampling can be used to approximate random sampling when used with microarchitectural simulation. LiveSim also uses systematic sampling to approximate random sampling during the setup phase.

During the setup phase LiveSim runs a fast emulation-only process that periodically forks copies of itself to create an in-memory set of checkpoints that contain all the architecturally visible state necessary to continue benchmark execution. Figure 2 illustrates how these newly created checkpoints enter a wait mode listening for commands to start microarchitectural simulation. Since forking a process uses copy-on-write, the checkpoint creation step is relatively cheap during the setup phase. However, in our implementation each checkpoint uses tens of megabytes which makes its memory consumption worthy of study. In Section 5.4 we evaluate checkpoint size and how to determine the optimal number of them.

In addition to the architecturally visible state, a checkpoint also needs a way to warm up the microarchitectural state before collecting statistics from a sample. In our experiments we found LiveSim was able to warm up most of the microarchitectural state, including an advanced O-GHEL branch predictor, with only 1 million instructions of warmup. However, effectively warming up the cache required more than 50 million instructions on average, and research indicates larger caches may require even more warmup [15]. Executing this many

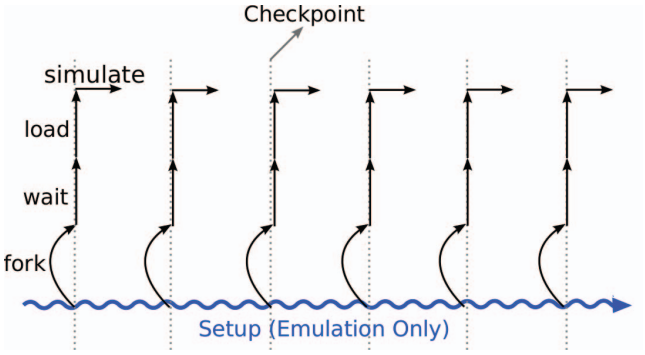


Figure 2: During setup the benchmark is emulated and checkpoints are created by periodically forking new processes. Each checkpoint process enters a waiting mode. Once LiveSim starts, each checkpoint loads the simulation dynamic library and configurations; then it starts simulation.

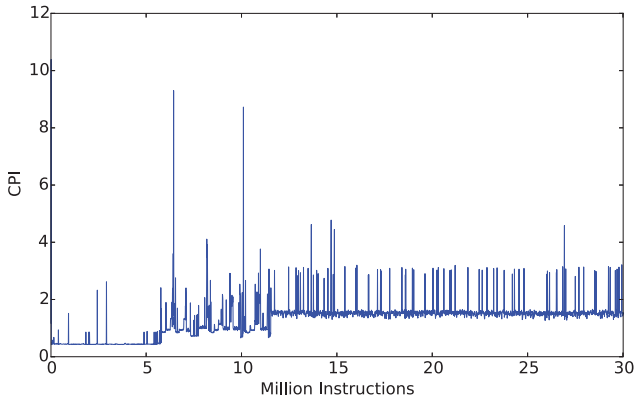


Figure 3: The CPI trace of Astar benchmark in SPEC 2006. The infrequent but extremely large spikes will have a considerable affect on the average CPI. Missing these spikes in random sampling will result in an unreliable sample mean.

instructions to warm up a sample would make the simulation too slow to meet the 5 second near real-time targets for LiveSim.

To solve the cache warmup problem we adapted the memory timestamp record (MTR) technique proposed by Barr et al. [14], and we call our adapted cache warmup technique LiveCache. LiveCache is implemented as a very large and highly associative cache that is larger than the largest cache that will ever be simulated. Each cache line in LiveCache has a counter field which stores a timestamp of the most recent access to this line. Each memory operation increases this timestamp and stores it in the counter field of the cache line it is accessing. These counters provide an ordering of all locations that could possibly be in the cache. Maintaining this state has a low overhead and does not slow down the LiveSim setup process very much. It will automatically be made available to the newly spawned checkpoint when they are forked, and it is independent of the microarchitecture that will be simulated later.

When LiveSim starts simulating a sample from a checkpoint, it first loads the simulator's dynamic libraries and configuration information. Next LiveSim executes all memory operations saved in the checkpoint's LiveCache in least recently used order without advancing the clock or collecting any statistics. This warms up the sample's microarchitectural cache state. Once all LiveCache memory operations are executed, the real simulation starts. This is similar to the technique proposed by Barr et al. [14] with some slight changes to simplify the integration with the LiveSim simulation infrastructure.

3.2 Calibration

The central limit theorem is the underlying foundation for the statistics theory that allows us to approximate the distribution of sample averages as though it were normally distributed. A typical rule of thumb is that 30 samples are enough to apply the central limit theorem when the population that is sampled is not highly skewed. But if the population is highly skewed then more samples are required.

Figure 3 shows a trace of CPI values for the Astar benchmark plotted over time for the first 30 billion instructions of the benchmark execution. This distribution is highly skewed for two reasons. First, the minimum CPI in the simulated 4-wide system is 0.25, but the maximum CPI is effectively unbounded and we can see spikes as high as 10 for this benchmark. Second, the spikes are relatively rare and most of the samples have a CPI much closer to 1. Simply using random sampling can require hundreds of samples for a distribution that is this highly skewed. Furthermore, while we can calculate the number of samples needed if we are given the population distribution, this information is not known a priori. Thus random sampling alone is unable to meet the execution time constraints of LiveSim.

However, LiveSim is able to take advantage of the correlation between code signatures and performance [16] and use this information to group the checkpoints into clusters. When results are calculated, LiveSim ensures that each cluster has at least 1 sample that is selected, and it also weights the results from each cluster based on the cluster size. This technique has some similarities to what Perelman et al. [9] do to statistically bound the error for results obtained using SimPoint.

LiveSim groups the checkpoints based on the performance statistics that are obtained with the baseline simulator configuration. Since performance statistics correlate with code signatures, this grouping tends to cluster the checkpoints together in a way where even radical changes in the simulated microarchitecture still cause the checkpoints to be clustered in a similar way. The clustering does not need to be exactly the same for different simulated microarchitectures, just close enough to avoid problems with extreme outliers that may otherwise skew the results.

After all of the checkpoints are spawned during setup

LiveSim begins calibration. For each checkpoint LiveSim loads the baseline simulator configuration and simulates a sample from the checkpoint. After all of the checkpoint samples are collected LiveSim uses a clustering algorithm to group the checkpoints into clusters.

For clustering LiveSim uses K-means algorithm where K ranges from $K = 1$ to $K = numcheckpoints/2$ and LiveSim attempts to find the value of K that provides the optimal trade-off between the variation of samples in clusters and the number of clusters (with the goal of minimizing both of these values). For each iteration of K LiveSim finds the best possible grouping of checkpoints to minimize the total variation of the metric of interest (typically CPI) across all K clusters. As LiveSim iterates through different values of K it keeps track of the value of K (and the associated configuration) seen thus far that minimizes total variation. When the algorithm finishes LiveSim uses the value of K that minimized total variation as the selected configuration for clustering checkpoints.

The final step in calibration is to assign weights to each cluster. This is done based on the number of checkpoints that are assigned to each cluster. For example if there were 2 clusters, and the first cluster had 900 checkpoints, while the second cluster had 100 checkpoints, then the first cluster would have a weight of 0.9 and the second cluster would have a weight of 0.1. These weights are used for averaging results obtained from simulated samples and reporting LiveSample results.

3.3 LiveSample

Once the setup and calibration phases are complete LiveSim is ready for interactive use. The usage scenarios that we envision is that the setup and calibration phases can be completed when the architect is not actively using the simulator (similar to how simulation batch jobs are run today). The LiveSample and LiveCI results are what the architect would be interested in seeing while using LiveSim for Live simulation. An architect may make a configuration change and then request results from LiveSim.

At this point LiveSim randomly selects the first batch of checkpoints to simulate. The selection algorithm depends on the number of clusters and the computation resources of the system used for running the simulation and is shown in Algorithm 1. The reason that we spawn at least twice as many checkpoints as cores in the host system is that it provided the highest sample execution throughput on our system. Too many running samples will overload the computation resources of the system, while too few limit opportunities for overlapping computation with I/O. This part of the initial checkpoint selection algorithm could be tuned differently for different systems, but it is important to ensure at least one checkpoint is executed from each cluster, regardless of the amount of samples that the system can execute in parallel.

The selected checkpoints are contacted by the LiveSim controller process and each selected checkpoint forks another copy of itself to run the simulation, while the parent checkpoint process goes back to its waiting mode. The child processes that will execute a sample loads the simulator library, initializes the cache state using the LiveCache data, warms up the rest of the microarchitectural state using detailed warmup, and finally collects statistics for its sample and reports them to the LiveSim controller process.

```
for all clusters do
    randomly select 1 checkpoint from the chosen
     cluster;
end
while num selected checkpoints < num cores * 2
 do
    randomly select 1 checkpoint from any cluster
end
```

Algorithm 1: Initial checkpoint selection algorithm

LiveSim begins reporting simulation statistics to the user as soon as the first checkpoint finishes execution. These statistics are calculated by computing the arithmetic mean of the sample values, after weighting each sample by its cluster weight. These results are what we call the LiveSample results and in our experiments they typically reached a steady state value within 5 seconds of starting the simulation.

3.4 LiveCI

Figure 1 shows an example of how LiveSim produces LiveSample and LiveCI results for a user. The LiveSample result is provided as soon as the first checkpoint is simulated, and it typically reaches a steady state value very quickly (typically within 5 seconds). However, the initial LiveCI results take slightly longer before they are available and the simulation continues running until the LiveCI result reaches the user's specified threshold.

When LiveSim selects checkpoints to use for LiveSample results it ensures that at least 1 sample is selected from each cluster. Afterwards it begins selecting checkpoints completely randomly. However, for calculating the confidence interval these samples that were initially selected from clusters cannot be used. The reason is that the confidence interval calculation requires samples to be chosen randomly from the population, and the clustering violates this requirement.

Consequently when LiveSim selects checkpoints for the LiveCI results it starts by selecting 30 completely random checkpoints to take samples from. If any of the checkpoints happen to have already been simulated then the earlier sample results can be used directly. Otherwise the LiveCI results have to wait until at least 30 completely random samples have been simulated. LiveSim requires a minimum of 30 samples before calculating the confidence interval because 30 is a generally accepted heuristic as the minimum cutoff value for applying the central limit theorem to assume that the sample mean distribution is normally distributed at this rate and we are allowed to calculate confidence interval.

However, the minimum value of 30 is simply a heuris-

tic, and in a highly skewed distribution it may not be enough. In some of our initial experiments we observed that this could result in the confidence interval being reported as more precise than it really was. This happened in cases with a population that was mostly homogeneous, but had a few large spikes (such as the Astar example shown in Figure 3). If the initial set of samples did not contain one of the spikes, the samples variance could be very small which would lead to a very tight confidence interval being calculated for a sample mean that did not match the true sample mean of the population. On the other hand if the initial set of samples did contain a spike the confidence interval would be very large and outside of the user defined range. In this case the simulation would continue running and eventually enough samples would be collected so that an accurate sample mean and confidence interval was calculated.

So the only problem with the 30 sample minimum heuristic was that sometimes the simulation could end earlier than it should have because LiveSim missed one of the infrequent spikes. To solve this problem we developed a heuristic where we inserted a synthetic sample in the sample set when calculating the confidence interval. The synthetic sample was created by computing the average of the samples collected thus far and multiplying this average by 10. Then the confidence interval was calculated using the true samples as well as the synthetic one. This heuristic solved the problem of ending the simulation too early due to missing extreme outlier values and it works well in practice for the SPEC CPU 2006 benchmarks that we simulated. In the event that a user was simulating a workload with even more extreme variation in sample metric, a different heuristic might be required, but we expect that adding a synthetic point that is 10 times the sample average should work well enough for most workloads that computer architects simulate.

4. MEASUREMENT SETUP

We evaluated LiveSim using the 24 of the 29 SPEC CPU 2006 benchmarks that we were able to run with our simulator. Our simulation infrastructure uses the MIPS64r6 ISA and it was missing support for some Fortran libraries which prevented us from running 5 of the CPU 2006 benchmarks (bwaves, gamess, cactusADM, leslie3d, wrf).

We ran all of the benchmarks on an x86 system with Haswell microarchitecture for 10 seconds using the first reference input set and used performance counters to determine how many instructions the benchmark executed during this time. We then rounded this up to the nearest 5 billion instructions and used this as the number of instructions to simulate during our evaluation. This ranged from 15 billion to 90 billion instructions depending on the benchmark.

We implemented LiveSim using a modified version of ESESC [17] as the timing simulator and a modified version of QEMU [18] as the emulation engine.

We compared 3 different simulated microarchitectures: a high performance (HP) configuration that was modeled on Intel's Haswell microarchitecture, a medium performance (MP) configuration modeled on the ARM A72 microarchitecture, and a low performance (LP) configuration modeled after MIPS Apache microarchitecture. Table 1 provides more details about the simulated microarchitecture configurations. The confidence level for all evaluations was set to 95% and the target confidence interval was 10% of the reported values.

Parameter	HP	MP	LP
Freq	3.5 GHz	2.5 GHz	1.7 GHz
I$	32KB 8w 2cyc	32KB 2w 2cyc	32KB 4w 2cyc
D$	32KB 8w 4cyc	32KB 4w 4cyc	32KB 4w 4cyc
L2	256KB 8w 11cyc	2 MB 16w 16cyc	1MB 8w 26cyc
L3 (shared)	8MB 16w 20cyc	N/A	2MB 32w 14cyc
Mem.	110 cyc	100 cyc	60 cyc
BPred.	ogehl 10*2K	Hybrid 16K	2level 2K
Issue	4	3	2
ROB	192	128	64
IWin.	60	72	64
LSQ	72/42	32/32	24/24
Reg(I/F)	168/168	128/128	40/40

Table 1: The three different architectures used to evaluate LiveSim.

Our host system had 2 Intel Xeon CPU E5-2689 CPUs and 192 GB of main memory. Each CPU had 8 cores with 2 SMT threads per core, giving a total of 32 OS visible logical processors. When running the benchmarks we executed them one at a time on the host system, which allowed the host to use all its resources for running a single benchmark. In all our evaluations, the simulated architecture is single-core and only one single threaded benchmark is running at a time.

5. EVALUATION

Our evaluation of LiveSim is focused on four different areas: speed, accuracy, warmup, and sample characterization. For speed and accuracy we compared LiveSim with no sampling simulation and with a sampling mode that was very similar to SMARTS [4].

5.1 Speed

Our primary goal for LiveSim is to enable interactive design space exploration using a microarchitectural simulator. To evaluate this we ran all 24 benchmarks for each of the 3 different configurations using both sampling and no sampling, and we recorded a time varying trace of the LiveSim results. We calculated the time varying CPI error percentage for each benchmark by comparing the benchmark CPI reported by LiveSim with the CPI simulated without sampling. Figure 4 shows a graphical representation of this data. The important thing to note is that although many of the initially reported values have a large CPI error, this quickly stabilizes and within 5 seconds the average error has dropped to 3.51%. Furthermore this error stays roughly the same over the next 5 seconds. As a result we believe that the LiveSample results reported after 5 seconds of simulation make LiveSim suitable for interactive microarchitectural simulation. This is even more impressive considering that the portion of the bench-

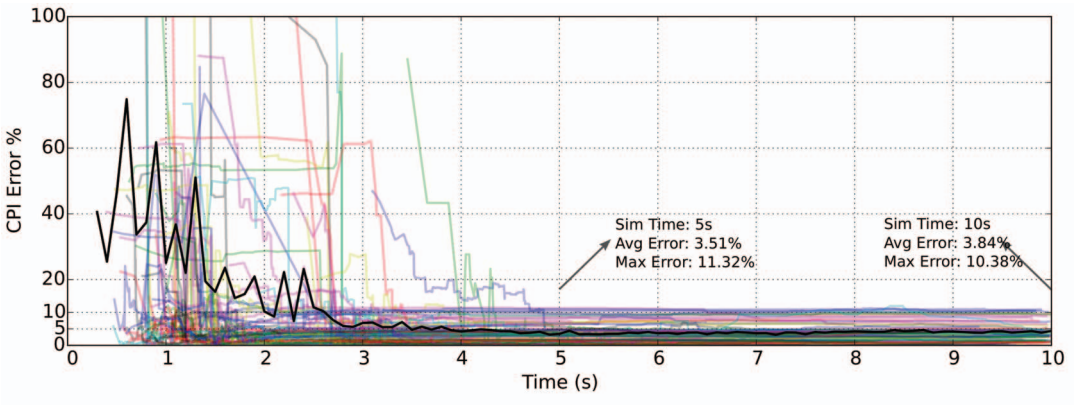


Figure 4: LiveSim CPI error for all 3 configurations and all 24 benchmarks (black line shows average error). LiveSim achieves an average of 3.51% CPI error within 5 seconds.

mark simulated is equivalent to 10 seconds of execution on a high performance system with a Haswell microarchitecture. This means that LiveSim enables simulation that is even faster than native execution.

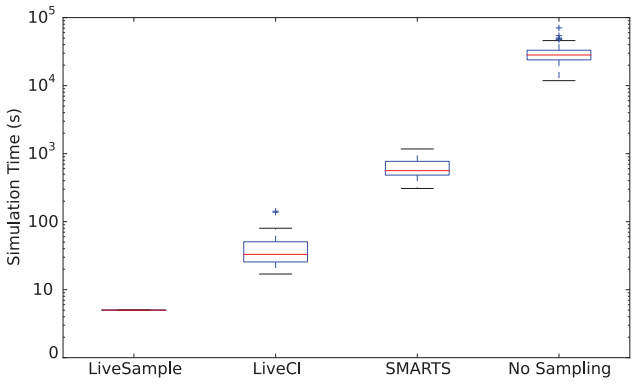


Figure 5: Average simulation time of all benchmarks and configurations. LiveSample results are ready within 5 seconds, LiveCI takes tens of seconds, SMARTS takes tens of minutes, and running without sampling takes many hours.

With LiveSim we define LiveSample as the initial results that are produced using a weighted average of samples from the checkpoint clusters created during the calibration step. The results in Figure 4 indicate that the LiveSample results are usable within 5 seconds. However, the LiveCI results take longer because they require true random selection of a larger number of samples. Figure 5 shows a comparison of runtime for all of the different configurations for LiveSample, LiveCI, SMARTS, and running the simulation without sampling. It is important to note that the execution time of SMARTS and no sampling is proportional to the length of the benchmark while the execution time for LiveSample is nearly constant and for LiveCI it is proportional to the variability of the samples. Hence, simulating a longer benchmark won't necessarily increase the simulation time for either LiveSample or LiveCI.

Table 2 provides additional insight about the speed of LiveSample and LiveCI and how it varies per benchmark using the HP configuration. The execution time of LiveCI is proportional to the number of samples that need to be simulated, and this is typically proportional to the variability of the benchmarks. For LiveSample we don't show a specific time since it is not determined algorithmically. But as illustrated earlier it is typically stable within 5 seconds. The execution time of LiveCI is determined algorithmically and it varies quite a bit from one benchmark to another. However, for most benchmarks it finishes within a minute or less. The MP and LP configurations typically finish running more quickly than the HP configuration, which is why the overall average execution time for LiveCI of all benchmarks and all configurations is 41 seconds.

Setup and calibration only need to be performed once for each benchmark even when there are big changes in configurations (e.g. LP to HP) or even code changes in simulator. In our experiments the average setup time was 18 minutes and the average calibration time was 3 minutes. The only steps taken in every simulation are LiveCache reload which takes 0.6 seconds on average, branch predictor warmup which takes 1 second, LiveSample timing which takes 3.4 seconds and LiveCI which can add 16 to 132 additional seconds to the simulation time.

5.2 Accuracy

In the previous section we demonstrated that LiveSim is fast enough to be used for interactive simulator use. However, fast results are only useful if they are reasonably accurate. When evaluating accuracy there are two things to consider: how close is the point estimate to the true value, and how often is the true value within the confidence interval. For our evaluation we selected a confidence interval of 10% and a confidence level of 95%. This means that we expect at most 5% of the simulation results to vary by more than 10% from the true value.

To evaluate this we ran 9 different experiments for each of the 24 benchmarks. In each experiment, we

Benchmark	LiveSample	LiveCI	LiveCI Time (s)
astar	10	433	67.163
bzip2	10	496	64.021
calculix	6	398	48.227
dealII	1	178	17.937
gcc	14	490	80.032
gemsfdtd	6	328	45.724
gobmk	4	360	56.750
gromacs	2	246	30.327
h264ref	1	178	24.756
hmmer	1	177	21.247
lbm	3	239	32.679
libquantum	1	194	33.360
mcf	13	321	64.381
milc	13	500	74.741
namd	1	180	22.842
omnetpp	4	185	33.932
perlbench	6	183	33.197
povray	1	177	21.083
sjeng	2	899	137.470
soplex	9	427	61.676
sphinx3	8	213	27.493
tono	4	190	21.105
xalancbmk	1	177	31.461
zeusmp	8	412	54.872
Average	**5.4**	**315.9**	**54.8**

Table 2: Number of checkpoints needed for LiveSample and LiveCI as well as LiveCI execution time for each benchmark.

first ran calibration with one configuration (HP, MP, or LP) and then ran LiveCI with another configuration (we tested all possible combinations). Of the total 216 experiments, there were 9 instances where the true CPI value was outside of the range reported by LiveCI. This is 4.16% of the time, which is within the expected range for a 95% confidence level. Figure 6 shows the distribution of CPI error for this set of experiments. Although the target confidence interval was set to 10%, in most cases the actual CPI error was much less, and the overall average error was 3.39%. These results also support our contention that the calibration step is microarchitecture independent. The overall error is roughly the same regardless of whether calibration is done with the same configuration as LiveCI or if the configuration used for LiveCI is very different from that used for calibration.

Figure 7 compares the CPI from LiveCI and full simulation when using the same configuration for calibration and LiveCI. In this case all of the benchmarks are within the configured confidence interval.

5.3 Warmup

Earlier we described how architectural state warmup is a critical part of sampling. In this section we evaluate the effectiveness of LiveCache as well as the amount of detailed warmup for other parts of the simulated microarchitecture. All of these experiments were performed using the HP configuration since it has the largest and most complex microarchitectural structure that will need the most warmup.

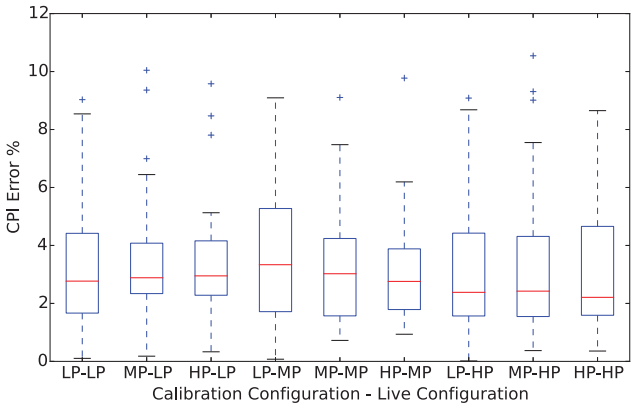


Figure 6: CPI error distribution across benchmarks in LiveSim. Each box label shows calibration and live simulation configurations respectively.

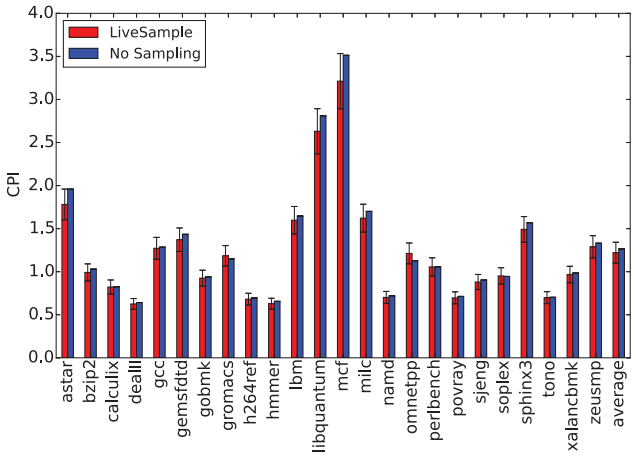
LiveCache is created to eliminate the need of doing traditional cache warmup for millions of instructions. In order to see how it compares, we ran a set of experiments with different amounts of warmup varying from 0 to 102.4 million instructions. In each experiment we measured and recorded the AMAT error for all benchmarks. Figure 8 shows the result of these experiments comparing them to one other experiment in which LiveCache was enabled and no other warmup was done. Our evaluations show that having LiveCache enabled results in less AMAT error in average than doing 50m instructions warmup and less maximum AMAT error than 102m instructions warmup. We also measured the overhead associated with LiveCache per checkpoint and it was less than 0.6 seconds on average. This is a relatively small overhead compared to the sample execution time, and is less significant because samples are executed in parallel.

Although LiveCache eliminates the need for cache warmup, there are other microarchitectural components that need to be warmed up especially the branch predictor. Figure 9 shows how branch prediction accuracy error rate decreases as the amount of warmup increases. It hits a plateau around 900K instructions, and we found that in practice 1 million instructions was a good value for warmup.

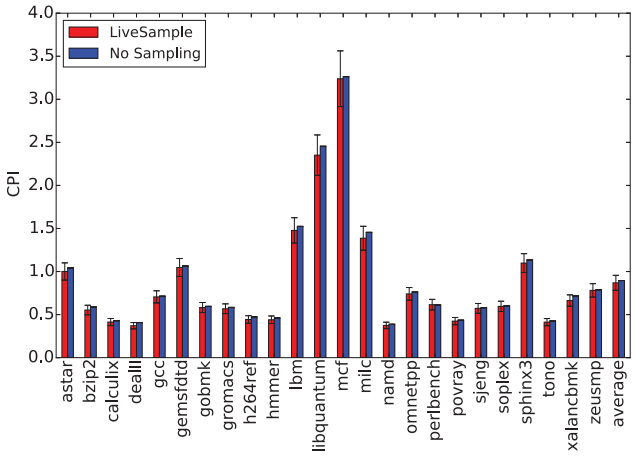
In our evaluations, we used ESESC [17], which has a typical timing simulation speed of 1 MIPS. Since ESESC is fast it does not need a separate functional warmup for branch predictor. However, a slower simulator may need to have a separate branch predictor warmup mode.

5.4 Checkpoint Characterization

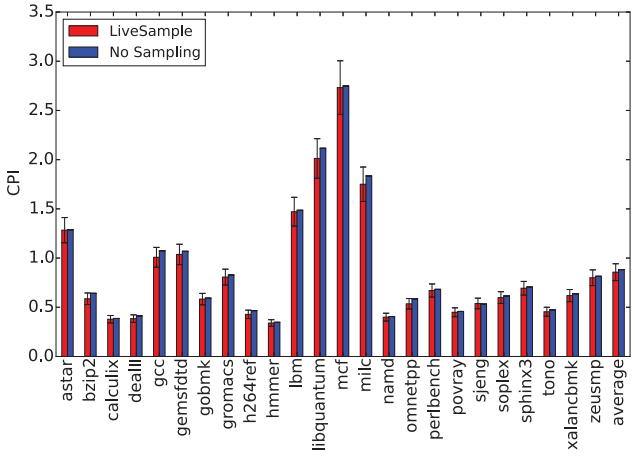
Although the number of samples that LiveSim uses is determined on-the-fly, the maximum value is limited by the number of available checkpoints. As a result we need to make sure to create enough checkpoints for any possible combination of configuration and benchmark



(a) Low Performance



(b) Medium Performance



(c) High Performance

Figure 7: LiveCI results of SPEC benchmarks simulating the LP, MP and HP architectures compared to no-sampling simulation. The reported CPI results have 3.33% error in average and CI estimation is 100% accurate.

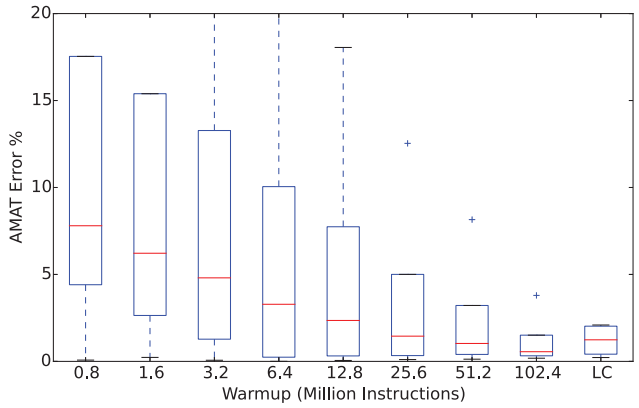


Figure 8: Comparison of AMAT error for LiveCache(LC) and traditional cache warmup.

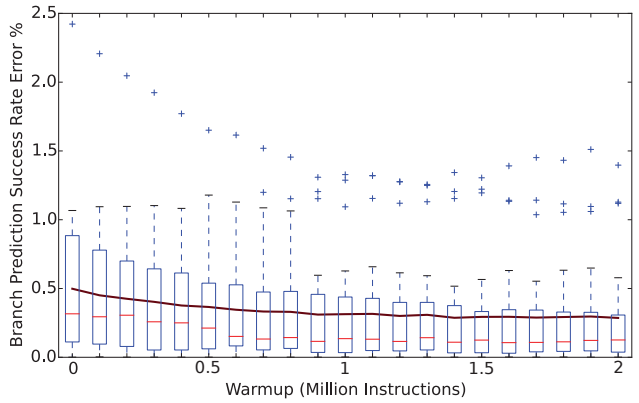


Figure 9: Average error of branch prediction statistics based on amount of detailed warmup at the start of a checkpoint.

that might be simulated. On the other hand, there are two reasons why we want to limit the number of checkpoints, although we think these problems are more significant in simulator development than they would be if LiveSim were used in practice. The first reason is that each checkpoint needs to be run during the calibration phase, so adding more checkpoints makes calibration take longer. (However, if LiveSim were used in practice we expect that calibration would be done infrequently relative to how often the user collected LiveSim and LiveSample results, and longer calibration times would not be a problem.) The second reason is that our current implementation potentially uses a large amount of memory for each checkpoint, and if LiveSim has to use swap it will have a dramatic performance drop. We believe that this is mostly an implementation issue rather than something that is intrinsic to the LiveSim system, and that memory use per checkpoint could be reduced if more time were spent optimizing this bottleneck.

LiveSim uses copy-on-write when creating checkpoints so the memory utilization depends on how many pages the application writes to. We have measured the mem-

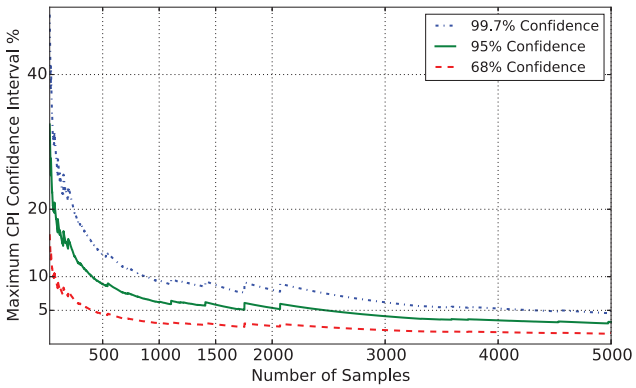


Figure 10: Maximum number of samples needed for a given confidence interval and confidence level in LiveSim, estimated using Monte Carlo simulation.

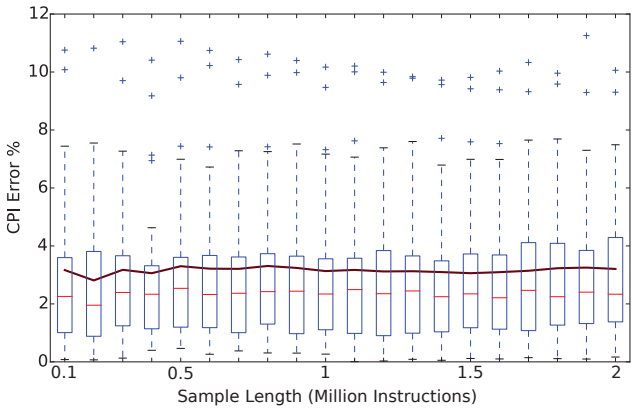


Figure 11: The effect of sample size on CPI error. Each box shows the error rate distribution for SPEC benchmarks and the line shows the average error across benchmarks.

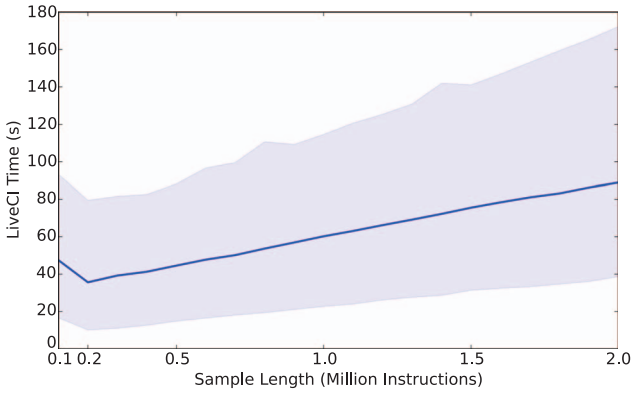


Figure 12: The effect of sample size on LiveCI time. The line shows the average LiveCI time across benchmarks and the area is where the actual distribution lies.

ory usage per checkpoint for SPEC benchmarks, and on average, a new checkpoint adds 45 MB memory occupancy. The maximum checkpoint size belongs to MCF, which is 165 MB.

To determine the maximum number of checkpoints that might be needed for various confidence interval and confidence level targets we ran the simulations without sampling and collected samples for every possible checkpoint candidate. This gave us a pool of tens of thousands of potential samples to pick from. Next we ran a Monte Carlo simulation to randomly select from the pool of samples. For each benchmark and configuration pair, we calculated the confidence interval from the set of samples, and saved the maximum confidence interval calculated for that number of samples. We did this for the three most common confidence levels, and Figure 10 shows a plot of the results. The plot indicates that for our target of 10% confidence interval at a 95% confidence level we need roughly 500 checkpoints. However, this is simply a heuristic and to be safe we recommend doubling the number shown here when picking how many checkpoints to actually use, because if LiveSim does not have enough checkpoints it may be unable to meet the confidence interval target for LiveCI results.

We also evaluated how many instructions each sample should contain and the impact of sample size on simulation error and runtime. In general larger samples tend to improve accuracy but decrease simulation speed. We determined that there is a sweet spot where increasing sample size does not improve accuracy but does decrease the speed. Furthermore in our implementation we did not get any speedup for samples that were smaller than 100K instructions because of communication overhead between the simulation server and the worker nodes. We experimented with sample sizes ranging from 100K instructions to 20 times that amount. Figure 11 shows that the average error does not decrease with larger samples sizes.

Although Figure 11 indicates that the minimum sample size is best in terms of speed and accuracy trade-off, this is not necessarily the case, because smaller samples can have more variation, and more variation across samples increases the number of samples needed for LiveCI. Figure 12 shows the effect of sample size on simulation time (LiveCI). This figure shows that 200K instruction samples result in the minimum simulation time. Since the error rate is nearly the same for all samples sizes, this is the sample size we used for LiveSim.

6. RELATED WORK

Researchers have been working on ways to speed up simulation for decades and we surveyed some of the seminal work related to profile based sampling [3, 9] and statistical sampling [4, 10, 11] for microarchitecture simulation in Section 2. To the best of our knowledge no one has proposed simulation techniques that are suitable for interactive use (providing results in 5 seconds or less). LiveSim achieves fast simulation by combining three main techniques: random sampling of

checkpoints, parallel simulation of checkpoints, and fast warmup of checkpoint state using LiveCache. There is a variety of related work in these various areas, but none of them attempt to achieve the goals of LiveSim.

The most closely related work to LiveSim is from Sandberg et al. [19, 20]. Like us, they use copy on write to fork multiple checkpoints and execute the checkpoints in parallel to speed up simulation. However, their proposal focuses on accelerating a single simulation run and only executes at 25% of native execution speed when simulating a system with an 8MB L2 cache. While this is an impressive result, our LiveSim system is able to execute at faster than native speed. After the initial setup step, LiveSim is able to provide simulation results in 5 seconds or less, even though we simulated 10 seconds of native execution. Sandberg et al. essentially use the SMARTS methodology, while using virtualization and parallel checkpoint execution to accelerate the function warming (which is the most time-consuming part of SMARTS). In contrast we randomly select checkpoints in LiveSim and are able to report initial results within 5 seconds, and we choose how many total checkpoints to execute based on characteristics of the benchmark that we are simulating, whereas Sandberg et al. simulate all checkpoints as SMARTS would. Parallel execution of forked copies of an application has also been used by others to speed up analysis performed using dynamic binary instrumentation [21, 22].

SMARTS is effective at minimizing the number of instructions that need detailed simulation; however, its conservative always-on warmup of caches and branch predictor makes warmup the simulation bottleneck (over 99% of simulation time). Many researchers have observed that always-on warmup of caches may be unnecessary and have looked for ways to accelerate warmup. For LiveSim we developed LiveCache by adapting a technique developed by Barr et al. [14] which keeps track of the sequence of memory operations during functional warmup and uses this information to rebuild the cache state before beginning detailed simulation of a sampling unit. We found that LiveCache technique works very well with LiveSim and helps us meet our goal of getting accurate simulation results in 5 seconds or less. However, there are a variety of other techniques that have been proposed for accelerating warmup.

Haskins and Skadron [23, 24] demonstrated that continuous cache and branch predictor warmup was unnecessary, and they proposed ways to determine when to begin warmup prior to simulating a sample. Eeckhout et al. [25] proposed a similar technique that further reduced the amount of warmup required. Luo [26] proposed a method to monitor when a cache was warmed up and used that information to decide when to switch to full simulation. Recent work from Nikoleris et al. [15] shows that some workloads may require up-to 100 million instructions of cache warmup for caches larger than 64 MB. They propose a technique that uses native execution to capture a sample of memory accesses and uses this to reduce the amount of warmup for large caches. All of these techniques are effective for the types of simulation they evaluate, but they would not help with LiveSim because we still need to execute the application once during the setup phase, and so LiveCache is easily integrated with LiveSim's setup phase as a low overhead and relatively simple way to do cache warmup.

For LiveSim we have focused on developing a simulator that supports interactive use when evaluating new architecture proposals. Our work focuses on fast performance simulation for a single thread of execution because this is the baseline for microarchitecture simulation, and it must work correctly before considering more complex scenarios. Other researchers have looked for ways to speed up thermal simulation [27, 28], multithreaded simulation [17, 29, 30], and simulation of soft-errors in caches [31]. As future work we may extend LiveSim to support these additional simulation modes, but first we want to establish the usefulness of LiveSim using performance simulation only.

There are also proposals to accelerate simulation by varying the level of simulation detail depending on the region of code that is being simulated [32, 33, 34]. While these techniques work well for accelerating simulation they fall short of our goal of supporting simulation speeds that are suitable for interactive use.

7. CONCLUSION

We developed LiveSim, a novel simulation methodology that can be used for interactive microarchitectural design space exploration. Although analytical modeling can also be used for early design space exploration, eventually architects typically use simulation based methods to evaluate the usefulness of proposed ideas. LiveSim makes simulation fast enough for interactive use and allows architects to quickly change parameters and get immediate feedback using real benchmarks. LiveSim leverages many advances of the past two decades in applying statistical sampling to microarchitectural simulation. However, previous work on sampling has simply tried to make simulation faster. LiveSim is the first to demonstrate how sampling can be used to support interactive microarchitectural simulation.

Our prototype demonstrates the feasibility of LiveSim and obtains accurate results within 5 seconds and bounds the possible error within 41 seconds on average for the benchmarks we evaluated. It is available as an open source project at https://github.com/masc-ucsc/liveos. Although we evaluated the LiveSim methodology using this prototype, the concepts are general and can be adopted for use with other simulators.

Acknowledgments

We thank the anonymous reviewers for their feedback on the paper. Special thanks to Ethan Papp and Alejandro Aguilar for their help developing the LiveSim software. This work was supported in part by the National Science Foundation under grants CNS-1059442-003, CNS-1318943-001, and CCF-1337278. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

8. REFERENCES

[1] A. J. KleinOsowski and D. J. Lilja, "Minnespec: A new spec benchmark workload for simulation-based computer architecture research," *IEEE Computer Architecture Letters*, Jan. 2002.

[2] J. Wawrzynek, D. Patterson, M. Oskin, S.-L. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic, "Ramp: Research accelerator for multiple processors," *IEEE Micro*, vol. 27, no. 2, March 2007.

[3] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Oct. 2002.

[4] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *International Symposium on Computer Architecture (ISCA)*, Jun. 2003.

[5] W. C. Hsu, H. Chen, P. C. Yew, and H. Chen, "On the predictability of program behavior using different input data sets," in *Workshop on Interaction between Compilers and Computer Architectures (INTERACT)*, Feb. 2002.

[6] E. S. Chung, J. C. Hoe, and B. Falsafi, "ProtoFlex: Co-simulation for component-wise FPGA emulator development," in *Proceedings of the Workshop on Architecture Research Using FPGA Platforms (WARFP)*, Feb. 2006.

[7] D. Chiou, D. Sunwoo, J. Kim, N. Patil, W. Reinhart, D. Johnson, J. Keefe, and H. Angepat, "Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators," in *International Symposium on Microarchitecture (MICRO)*, Dec. 2007.

[8] M. Yourst, "Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator," in *International Symposium on Performance Analysis of Systems Software (ISPASS)*, Apr. 2007.

[9] E. Perelman, G. Hamerly, and B. Calder, "Picking statistically valid and early simulation points," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sep. 2003.

[10] T. Conte, M. Hirsch, and K. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," in *International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*, Oct. 1996.

[11] T. Conte, M. Hirsch, and W.-M. Hwu, "Combining trace sampling with single pass methods for efficient cache simulation," *IEEE Transactions on Computers*, vol. 47, no. 6, Jun. 1998.

[12] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "Turbosmarts: Accurate microarchitecture simulation sampling in minutes," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2005.

[13] T. Wenisch, R. Wunderlich, B. Falsafi, and J. Hoe, "Simulation sampling with live-points," in *International Symposium on Performance Analysis of Systems Software (ISPASS)*, Mar. 2006.

[14] K. Barr, H. Pan, M. Zhang, and K. Asanovic, "Accelerating multiprocessor simulation with a memory timestamp record," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2005.

[15] N. Nikoleris, D. Eklov, and E. Hagersten, "Extending statistical cache models to support detailed pipeline simulators," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2014.

[16] J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder, "The strong correlation between code signatures and performance," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2005.

[17] E. Ardestani and J. Renau, "ESESC: A fast multicore simulator using time-based sampling," in *International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2013.

[18] F. Bellard, "Qemu, a fast and portable dynamic translator," in *USENIX Annual Technical Conference*, Apr. 2005.

[19] A. Sandberg, N. Nikoleris, T. Carlson, E. Hagersten, S. Kaxiras, and D. Black-Schaffer, "Full speed ahead: Detailed architectural simulation at near-native speed," in *IEEE International Symposium on Workload Characterization (IISWC)*, Oct. 2015.

[20] A. Sandberg, "Understanding multicore performance : Efficient memory system modeling and simulation," Ph.D. dissertation, 2014.

[21] S. Wallace and K. Hazelwood, "Superpin: Parallelizing dynamic instrumentation for real-time performance," in *International Symposium on Code Generation and Optimization (CGO)*, Mar. 2007.

[22] T. Moseley, A. Shye, V. Reddi, D. Grunwald, and R. Peri, "Shadow profiling: Hiding instrumentation costs with parallelism," in *International Symposium on Code Generation and Optimization (CG))*, Mar. 2007.

[23] J. Haskins, J.W. and K. Skadron, "Minimal subset evaluation: rapid warm-up for simulated hardware state," in *International Conference on Computer Design (ICCD)*, Sep. 2001.

[24] J. W. Haskins Jr and K. Skadron, "Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation," in *International Symposium on Performance Analysis of Systems Software (ISPASS)*, Mar. 2003.

[25] L. Eeckhout, Y. Luo, K. De Bosschere, and L. K. John, "BLRL: Accurate and efficient warmup for sampled processor simulation," *The Computer Journal*, vol. 48, no. 4, 2005.

[26] Y. Luo, L. John, and L. Eeckhout, "Self-monitored adaptive cache warm-up for microprocessor simulation," in *Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct. 2004.

[27] A. K. Coskun, R. Strong, D. M. Tullsen, and T. Simunic Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2009.

[28] E. Ardestani, E. Ebrahimi, G. Southern, and J. Renau, "Thermal-aware sampling in architectural simulation," in *International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2012.

[29] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sampled simulation of multi-threaded applications," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2013.

[30] T. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout, "Barrierpoint: Sampled simulation of multi-threaded applications," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2014.

[31] J. Suh, M. Annavaram, and M. Dubois, "Phys: Profiled-hybrid sampling for soft error reliability benchmarking," in *International Conference on Dependable Systems and Networks (DSN)*, Jun. 2013.

[32] M. Ekman and P. Stenstrom, "Enhancing multiprocessor architecture simulation speed using matched-pair comparison," in *International Symposium on Performance Analysis of Systems Software (ISPASS)*, Mar. 2005.

[33] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2011.

[34] D. Genbrugge, S. Eyerman, and L. Eeckhout, "Interval simulation: Raising the level of abstraction in architectural simulation," in *International Symposium on High Performance Computer Architecture (HPCA)*, Jan. 2010.