A Comparison of the Effect of Warm-up Techniques on the Speed and Accuracy of Simulating Using Shorter SimPoints.

Motivations

Investigation into new computer-architectural features relies heavily on cycle-accurate simulation tools such as SimpleScalar [1], the use of which, in order to deliver the quality of information desired, is compute intensive and time-consuming. At the heart of most approaches to speed up simulation is a reduction in the portion of executed code to be simulated in such high detail. However, as the length of any given sequence of fully simulated instructions decreases, the impact of stale or erroneous micro-architectural state on any results garnered from that section of the simulation can increase.

When the SimPoint technique was first introduced, the expected contiguous block size for simulation was 100 million instructions, and, as any cold start effects would be amortized over said instructions, has never been significant concern as to the method used, if any, to warm up the simulation state. However, as contiguous block sizes in more recent applications of SimPoint-based approaches drop to 10 million and to 1 million instructions, it becomes more important to consider what degree of warm-up of the micro-architectural state is necessary to avoid introducing error [2], and the relative effects of warm-up techniques on simulation speed. While many of the potential warm-up techniques for simulating a set of SimPoints are both simple and well known, we do not believe there to have been previous work surveying them in direct comparison to each other, providing us with an opportunity for inquiry.

Related Work

In “How to Use SimPoint to Pick Simulation Points” [3], several classes of potential warmup techniques are briefly mentioned for use with SimPoint, namely: none, assume hit, stale state, calculated warmup, and continuous warming. However, while some qualitative statements are made concerning relative efficacy, no quantitative data is presented. We examine implementations from each of the above categories.

Haskins and Skadron introduced Memory Reference Reuse Latency [4] as a technique to determine the appropriate start points for calculated warming. While we initially planned on utilizing the publicly available extensions to SimpleScalar that implement their algorithms and on comparing results to other warming techniques, especially to fixed duration calculated warming, this line of inquiry was aborted on account of time constraints and left for future work. Numbers directly from [4] could not be used due to poor overlap of chosen benchmarks. MRRL focused on measuring the number of instructions that elapse between memory references in order to determine when warm-up should begin. The closer memory references that precede a sample will be more highly associated with the sample itself than references farther away. Thus, those closer references should be included in the warm-up while the references farther away have no need to be included because they are likely irrelevant.
DiST [5] presented a dynamic run-time warming mechanism. This method proposes using instructions immediately following a simulated section. The number of instructions for warm-up used is dynamically determined by comparing simulation results with the next section to be simulated. When they are both similar enough, warm-up is considered complete. The significant modifications we saw as necessary for this to be implemented in SimpleScalar caused efforts to implement this technique in our framework to be left to future work.

The SMARTS approach [6], utilizing much smaller contiguous detailed runs than we are investigating, showed that continuous warming, combined with pipeline warming immediately prior to simulation, could provide high accuracy even for very short samples. More important as a motivator for our work, however, was the attention paid to time spent fast forwarding through functional simulation as the potentially dominant overhead in simulation time, and the general focus on the importance of trading off simulation time and accuracy.

Methodology

For our simulation purposes, we modified Simple-Scalar 3.0 to support various warm-up techniques and combinations thereof. In order to be able to verify our results against known values, we ran a subset of the SPEC2000 benchmark alpha binaries, using the same base configuration as that used to produce values for the posted SimPoint error rates, namely:

<table>
<thead>
<tr>
<th>Queues:</th>
<th>Fetch: 32, Load-Store:32, RUU: 128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Predictor</td>
<td>Bimodal+2 level local predictor combination, each with 8K tables, chooser table also 8K</td>
</tr>
<tr>
<td>Memory latency</td>
<td>151 cycles</td>
</tr>
<tr>
<td>IL1-cache</td>
<td>8K 2-way 32 byte lines</td>
</tr>
<tr>
<td>DL1-cache</td>
<td>16K 4-way 32 byte lines</td>
</tr>
<tr>
<td>UL2-cache</td>
<td>1M 4-way 64 byte lines</td>
</tr>
</tbody>
</table>

While we had initially hoped to address SimPoints of both one and ten million instruction duration, time constraints forced us to leave investigation of the one million instruction duration points to a later study. Likewise, in the interest of time, as governed by available compute facilities, we did not attempt to gather MRRL data or to implement something akin to the DiST approach.

We chose to focus on the following set of warming techniques:

None – as a baseline, we simulated using no warm-up techniques at all. Absent such data, it would be impossible to even partially decouple the error rate reductions from any particular warm-up technique from error intrinsic to executing only part of the program via the SimPoint model.
Assuming hits on "cold-start" misses to the cache and branch prediction apparatuses - The "assume hit" warm-up method masks cold start effects by adding a warm-up bit to each block of the 3 caches (DL1, IL1, UL2), and for each entry in the branch predictor and BTB tables. This warm-up bit is set initially and is cleared once an entry in a cache or branch predictor is used. For that first use, a hit or correct prediction is assumed based on a configuration specified hit rate. As the high accuracy of these structures makes the likelihood of a hit or correct prediction quite high, we conducted runs assuming a 100% hit rate and runs assuming a 95% hit rate.

Using stale state – this warm-up method is less proactive then the "assume hit" method. "Stale state" does not attempt to mask the lack of warm-up by falsifying the first access to a structure’s entries, but also does not alter any part of the micro-architectural state when performing an interleaved sequence of functional and cycle-accurate simulation sections, excepting the pipeline state, which must be drained in order to return to functional granularity simulation. Essentially, the state from previous simulation sections acts as a proxy warm-up for the current simulation section.

“Calculated warm-up” – A rigorous implementation of this method would have called for running a number of instructions immediately preceding the start of a simulation point to accomplish warming based on having generated some notion of a working set of architectural data such as branch addresses and accessed data. However, in the interest of simplicity, and as we intended to use the MRRL approach to highlight intelligent calculated warm-up techniques, we only implemented a “blind” calculated warm-up that, depending on configuration parameters, will execute some fixed number of instructions prior to each simulation point in cycle accurate detail.

“Continuously warm”/functional warming – While our implementation was simplistic, in that it made little attempt to optimize for speed, we did implement continuous warming via informing the caches of all memory accesses and updating the branch predictors and BTB on every simulated branch. As addressed in our results section, it turned out later that our implementation produced flawed statistics for the branch target buffer, branch predictors, and cache access records. However, IPC error rates appear sufficiently low that it seems likely to be a flaw in the recording of those specific statistics due to some failure to mask reporting of our updates during functional simulation, and not a flaw in the correctness of the fundamental implementation.

We also investigated what we thought to be logical combinations of the above approaches:

Functional + 10,000 instruction blind calculated warm-up: While it seemed unlikely to matter too much over the course of 10 million instructions, as the opportunity cost of doing a small calculated warm-up was negligible, and SMARTS had noted that pipeline warming could play its own part in the accuracy of admittedly much smaller simulated runs, we gathered data on functional runs augmented with minimal calculated warm-up.
Stale + calculated blind warm-up: Stale state, preserving any global data that was frequently accessed, and calculated warm-up, bringing into memory some portion of data currently being operated on, are a naturally complementary pairing. We examined both 10,000 and 1 million instruction blind warm-ups for this combination so as to be able to speak to the degree that increasing the size of the calculated warm-up affects the pairing.

Calculated warm-up + assumed hit on uninitialized: Similar to the above, this pairing relies on the assumption that data that is not brought in by the calculated warm up is likely to be globally persistent data. For these runs, we used a blind calculated warm-up of 1 million instructions and an assumed hit rate on uninitialized entries of 100%.

Stale + calculated blind warm-up + assumed hit on uninitialized: A combination of all of the compatible warming techniques we implemented, we simulated runs using 100% hit on uninitialized for both 10,000 and 1 million instructions calculated blind warm-up.

While the published SimPoint error rates are for simulation runs using the {PC, # of occurrences} metric for when to begin simulation, given that we desired to be able to start cycle-accurate simulation in advance of reaching our target simpoint, we opted to continue to generate our runs using the instruction count metric for when to begin simulation. This is, no doubt, a source of additional error for our results.

Results

Due to time constraints, we were not able to run simulations for all of the benchmarks. Indeed, we were not able to complete simulation of all 12 combinations of warm-up approaches for those benchmarks we did address. Thus, there are some gaps present in some of our data, and some measurements could not be applied to some benchmarks.

Placing the warm-up data side-by-side, there are several immediately noticeable results. Figure 1 shows, for a set of benchmarks, the variance among normalized differences from the IPC of doing no warm-up at all (that is, variance over the set of deltaTechnique = (IPC_Technique-IPC_NoWarming)/IPC_NoWarming for all techniques that generated valid runs for that program-input pair). Examining figure 1, it is clear that certain program-input pairs are much more sensitive to warming than others are. Likewise, examining figures 2 through <FIXME>, it is clear that different parts of program execution differ greatly in their sensitivity to warmup.

It is also rapidly apparent that the usefulness of warm-up techniques is not level across benchmarks, nor is there a well defined ordering among the techniques we examined that transcends any particular benchmark. For gcc-expr and gcc-integrate, (Fig <FIXME>) it is useful to note not only the magnitude of the error, but also to discern between under and overshooting the true value. There appears to be an intrinsically over-optimistic slant to the SimPoints for these program-input pairs, and, as every warming technique except for not warming overshoots the actual IPC, the more aggressive warming techniques actually increase the error rate by causing higher IPC to be reported. This is in sharp contrast with such program-input pairs as gzip-log (Fig <FIXME>) where the best error
rate is associated with continuous warming. As the direction of error for the published SimPoint error rates, which, given what earlier published results used [2], one assumes to use perfect warming, are not given, it is difficult to see how well the direction of error correlates to the utility of more aggressive or intelligent warming techniques. However, as it is mentioned in [3] that the directional bias for various metrics is consistent across architectures, it seems entirely possible that the relative ordering of warming techniques by error could remain fixed across architectures as well.

We had hoped to use branch prediction and cache hit rates to gain deeper insights into why various warming techniques gave the results seen, but this proved less successful than desired. There does not appear to be a clear correlation between a particular deviation from the no-warmup case in terms of branch and cache behavior corresponding to a particular error rate. This is perhaps not surprising, given the small absolute errors involved. As can be seen in the figures in Appendix A, warming caused only very subtle changes to branch and cache rates for all of the gcc runs, yet gcc still exhibited noticeably varying error across the warming techniques. Likewise for gzip; the quite large fluxuations in I-cache miss rate the normalized graph seems to show are due to the I-cache miss rate being nearly zero; it too experienced notable variation in error rates among warming techniques without clear indicators appearing in branch or cache performance.

While in some cases (see Fig <FIXME>) the 10K instruction cycle-accurate warm-up notably altered the error rate of stale+10K relative to stale, the 10K additional instructions simulated, although of negligible time cost, had no discernable value. Stale + 1 million instructions of intro simulation was likewise more akin in accuracy to just stale warming than to warming just via execution of 1 million instructions. Clearly, 1 million was too small a number to use to see the full benefits of the approach via an unintelligent method, at the least for sets of 10 million instructions and quite possibly in an absolute sense. The composite warming techniques, however, performed quite well, tracking fairly closely to functional warming, albeit, due to the particulars of the very limited set of benchmarks we gathered data for, this did not always translate to having the lowest error rates.

Conclusions and Future work

While we fell far short of gathering all desired data, we were still able to glean some useful insights from the surveyed techniques. The degree to which continuous warming did not seem to offer consistent benefits was a somewhat surprising find, especially as it can be seen that the program-input pairs we reviewed were all at least somewhat sensitive to the choice of warming technique. If, as conjectured in the results section, which techniques produce lower error rates is a property that correlates in a manner similar to, or perhaps even dependent on, the directional bias of a set of simpoints towards various performance metrics, then it would seem worthwhile to determine, for each program-input pair, which family of warming techniques is best suited to providing a lower overall error rate. However, in order to determine the strength of the initial hypothesis, much more complete surveys would have to be made across different architectural
configurations and different simulation block sizes, and even then, it may cloud comparison between any two runs that did not use the same warm-up technique.

On a similar note, the variations in sensitivity to warming vs. non-warming displayed by the component simpoints of benchmarks we were able to get complete data sets for points directly to the success of approaches like MRRL which have highly variable behavior with respect to warming across a set of simulated blocks. It may be that, for optimal reduction of errors from warming, what is required amounts to having multiple warming approaches governed by some set of code properties that, taking intrinsic SimPoint bias into account, assign particular warming strategies to different portions of execution.

Despite not being optimized for speed, our abbreviated timing runs found that our continuous warming implementation was two to three times slower than our purely functional fast-forwarding implementation, which is not significantly more than reported for the difference between full and no warmup in [4]. We believe this to likely be because of a lack of speed optimization in our purely functional code. As our continuous warming code produced some erroneous statistics, we have omitted the actual timing runs, as they do not compare equally functional solutions. However, to a first approximation, it is sufficient to note that continuous warming takes significantly longer than the other techniques, and that (in no, only in, waiting for data... <FIXME> ) cases was it able to provide reductions in error commensurate with the additional overhead of continuous warming.

Aside from the time taken by continuous warming, we did not see any overbearing speed differences between simulation runs. This itself is notable, as some of these runs were doing 10% more cycle-accurate simulation than others. Clearly, time spent fast-forwarding through functional simulation is the dominant factor in overhead. This being said, it is important to note that, as we did not, as hindsight would lead us to conclude that perhaps we should have, make use of SimpleScalar’s checkpointing functionality, total compute time to generate the runs not making use of stale state was considerably higher, even in the presence of parallel facilities. However, even checkpointing has redundancies in time use due to having to create and load large files, and a more useful solution for doing a study such as this would have been to add a state-wiping function to our interleaved functional/cycle-accurate modified simulator so as to emulate having cold state. On a purely aesthetic note, we found managing the input and output organization of such single pass simulations can be significantly cleaner than generating a collection of configuration and output files which only together represent a simulation run.

As we underestimated the time necessary to achieve the breadth we desired our survey to have, there remains much work to be done. In particular, meaningful discussion of the tradeoffs between the added simulation speed overhead of keeping structures continuously warm and the accuracy gained from such a warming technique cannot commence until we have successfully re-implemented functional warming. Likewise, this survey is remarkably incomplete in terms of the number and type of benchmarks it has covered, and an extension of the inquiry across the rest of the SPEC benchmarks would lend significantly more confidence to any conclusions.

Even assuming that this survey were to be completed as originally intended for size 10 million instruction blocks, it is obvious even from the limited data gathered that doing a similar survey of size 1 million instruction blocks would yield much more pertinent data. Also, at the 1 million instruction block size, we would be well advised to use the PC and
occurrence number based method of determining when one has reached the beginning of a SimPoint, although doing so will require significant changes to our mechanisms for doing calculated warm-ups.


