Approximate Load Value Prediction: A Feasibility Study

Abstract

Approximate computing reduces the cost of computation, in time and/or power consumption, by allowing inaccurate results. The challenge is in controlling the effects of the introduced inaccuracy and providing guarantees of safety and quality of service. It has been shown that strict safety guarantees can be realized with programmer guidance, but providing strict quality of service guarantees remains an open problem. In this project, we evaluated potential benefits of Approximate Load Value Prediction (ALVP), an approximation technique that enables the processor to *bound* and *measure* every instance of the approximation with little overhead.

1. Introduction

In order to be effective, approximate computing systems need to control the effects of approximation. Two types of guarantees must be provided:

Safety. Any viable approximation technique needs to provide strict safety guarantees. Applying approximation should only cause graceful quality degradations without catastrophic failures such as segmentation faults or infinite loops.

Quality of service. In addition to safety, approximate computing systems need to provide at least a statistical guarantee of their quality of service. Providing general, strict quality of service guarantees remains an open problem.

Techniques for approximate computing generally introduce inexactness either directly through approximate operations (computations and storage) or indirectly by allowing occurrence of hardware-induced faults. Providing both types of guarantees mentioned above is significantly more difficult in case of the latter approach. When hardware faults are used as the source of inexactness, to ensure safety, the processor must be able to limit the faults to certain operations. For instance, it is usually expected that instructions are fetched and decoded reliably and control flow branch targets are computed correctly. Efficient implementation of such requirements is difficult because of power constraints and other practical issues. Providing quality of service guarantees is also more difficult due to the complexity of mapping hardware faults to errors in software.

Consequently, approximate computing techniques that explicitly introduce inexactness in the computation have been more widely explored in recent years. To ensure safety, a common practice is to rely on programmer annotations (sometimes with the assistance of static analysis methods) for identifying data and/or computations that are safe to approximate. The programming language (e.g. EnerJ [10]) then statically guarantees isolation of the precise program component from the approximate component through revised language semantics. Statistical quality of service guarantees are achieved either statically through profiling approximations (e.g. PetaBricks [2]) or dynamically via tuning approximation parameters (e.g. SAGE [9], Green [3], ApproxIt [13]).

SAGE [9] which is an example of systems that dynamically monitor and control quality of service works as follows: during offline compilation, it performs approximation optimizations on each kernel to create multiple versions with varying degrees of accuracy. At run-time, it tunes the parameters of the approximate kernels to identify configurations with high performance and a quality that satisfies the quality requirement. Since the behavior of approximate kernels may change during run-time, SAGE periodically performs a calibration to check the output quality and performance and updates the kernel configuration accordingly.

Such online approximation control schemes are generally more successful in optimizing the error-performance/energy tradeoff because of their adaptivity. However, their ability to monitor quality of service is limited due to their high *monitoring overhead.* Online measurement of the incurred error is essential to optimal tuning of the approximation configuration. But, each error measurement requires a redundant execution of the exact computation and incurs a substantial performance overhead of at least 50% (when approximation gain is 1 and computation of the quality of service takes negligible time). As a result, measurements are done only occasionally, leading to stale measurements and less-than-optimal approximation configurations.

2. Approximate Load Value Prediction

In this paper, we evaluate the potential benefits of a new approach for controlled approximate computing, Approximate Load Value Prediction (ALVP), that could achieve a more optimized error-performance tradeoff by enabling measurement and/or bounding of all approximation errors. The high-level idea of ALVP is to apply approximation to a form of speculative execution called load value prediction.

Load Value Prediction (LVP) leverages *value locality* to hide the memory latency of a cache miss. When a cache miss occurs, instead of waiting for the data, the predictor generates a value and allows the processor to continue executing instructions speculatively. When the data arrives, the prediction is validated against the actual value. If the predicted value is correct, the cache miss latency has been avoided. Otherwise, the processor must rollback the speculatively executed instructions. ALVP introduces approximation into LVP by relaxing its validation criteria. While in LVP, a prediction is validated only if it *exactly* matches the actual value, ALVP accepts predictions that are *close* to the actual value. This relaxed comparison can be realized simply by ignoring a number of the low-order bits, or *validation relaxation window*, of the comparison result and cheaply implemented using for example a mask register and a set of AND gates. This approximation scheme has some unique features that that differentiates it from previously discussed techniques:

1. The approximation error can be *bounded* even *before* sending the fetch request to the cache by selecting the width of the relaxation window. This can be used to tune the relative priority of approximation control and performance improvements. A wider relaxation window gives more priority to performance improvements by reducing the penalty paid for cache misses while increasing the probability of larger errors and lowering the quality of service. Exact operation can be enforced by setting width of the relaxation window to zero while the rollback mechanism can be bypassed by maximizing relaxation window width. A conservative but inexpensive way of implementing the bounding of the relative error incurred by ALVP is to only consider the most significant bit of the approximated value. If the predicted value, x, has its most significant bit at position b_m, a relaxation window of width w_r incurs at most a relative approximation error δ equal to:

$$\delta = \frac{\Delta x}{x} \leq \frac{2^{w_r} - 1}{2^{b_m}} \leq 2^{w_r - b_m}, \quad \text{when } b_m > w_r, \text{ and} \tag{1}$$

$$\delta = \frac{\Delta x}{x} \leq \frac{2^{w_r} - 1}{1} \leq 2^{w_r} - 1, \quad \text{when } 0 < b_m \leq w_r. \tag{2}$$

Eq. 1 gives a much tighter bound because the most significant bit is known. The bound in Eq. 2 is not very useful and would be typically avoided by choosing a narrower relaxation window. Relative error is not defined when the actual value is is zero ($b_m = 0$).

2. The error of *every* approximation instance can be *measured* upon the arrival of the missing cache block. This is in contrast to the software-based systems that sample quality of service values periodically using redundant computations and effectively eliminates the monitoring overhead. Error measurements are therefore kept up-to-date and may be used to control the quality of service. For example, approximation error of a load can be used to tune the confidence parameter of its predictor or to guide the selection of the relaxation window of a dependent load and control the accumulated error.

3. A Feasibility Analysis of Approximate Load Value Prediction

In this section, we evaluate the potential performance of an ALVP implementation. Optimal benefits of ALVP can be obtained by finding the tuning configuration that maximizes performance benefits while keeping quality of service in a specified acceptable range. We designed a two-phase search algorithm that tries to find this configuration. The two phases are described below.

Local optimization. In the first phase, load instructions are considered individually. A predictor is assumed for each load instruction. Each predictor is configured with all possible relaxation window widths and the number of mispredictions and approximation error bound are measured. A profiling pass is also performed in which the number of cache misses of each instruction is determined. Typical values of cache miss penalty (P_C) and value misprediction penalty (P_M) – both in number of clock cycles – are then used along with the misprediction (N_M) and cache miss (N_C) counts to estimate the number of *recovered cycles* according to Eq. 3, where misprediction penalty is subtracted from the cache miss penalty. Next, a score is assigned to each configuration according to Eq. 4. Here, δ is the relative error bound from Eq. 1 and 2. This metric, which was inspired by [11], calculates the harmonic mean of terms that estimate the performance increase and accuracy loss associated with the configuration. Finally, the load instructions are sorted according to the their maximum score across various relaxation window widths. In the next phase, only this best (according to the score) relaxation window width is considered for each load instruction.

$$N_R = N_C P_C - N_M P_M \tag{3}$$

$$score = \frac{2}{\frac{N_M}{N_R} + \frac{1}{1 - \delta N_M}} \tag{4}$$

Global optimization. In the second phase, the algorithm maintains a set of load instructions that can be approximated together without violating the quality of service requirement. At each step, the algorithm moves the topmost load instruction of the sorted list from the last phase to this set and measures the quality of service of the new tuning configuration. If the approximation violates the quality of service requirement, it is removed from the set and discarded. The algorithm terminates when the sorted list from the first phase becomes empty.

3.1. Implementation

We implemented the value predictor inside the data cache simulator of Pin [6] with a 64KB L1 data cache by adding the capability of monitoring and modifying accessed values. The simulator takes a tuning configuration and reports cache miss and value misprediction counts. The search algorithm was implemented in Python with the ability to feed the cache simulator with arbitrary tuning configurations, running the Pintool with different inputs and measuring the quality of service. We used a subset of the applications of PARSEC 3.0 [4] with the `simlarge` input set. For performance estimation, we assumed an L1 cache miss penalty of 12 cycles as is the case in Intel i7-4770 (Haswell) [1].

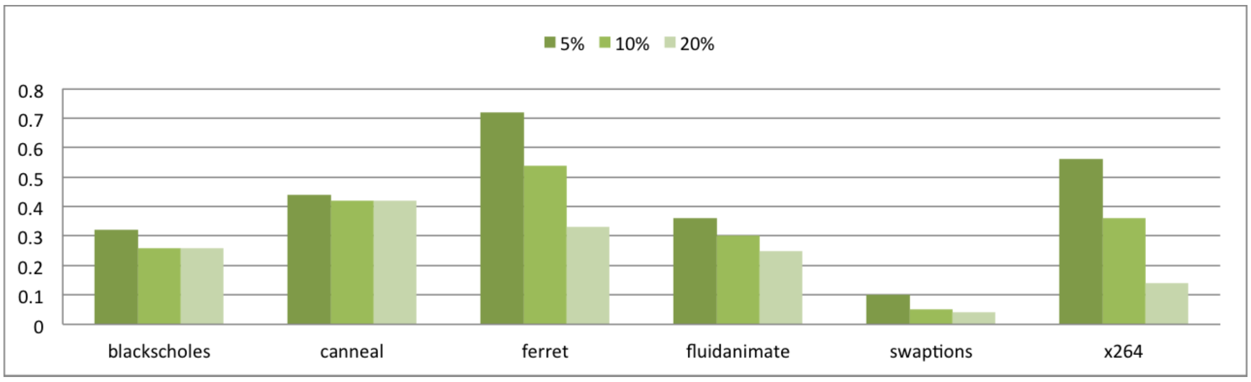
We assumed a recovery mechanism called *selective reissue* that is implemented in processors to recover in case instructions have been executed with incorrect operands. In particular, this is used to recover from L1 cache hit/miss mispredictions [5] (i.e. load dependent instructions are issued after predicting an L1 hit, but finally the load results in a L1 miss). Load value prediction follows this pattern as well. When the execution of an instruction with an incorrect operand is detected, the instruction as well as all its dependent chain of instructions are canceled and then replayed. A realistic estimation of the average misprediction penalty of selective reissue is 5-7 cycles [8]. We assumed a misprediction penalty of 7 cycles.

3.2. Results

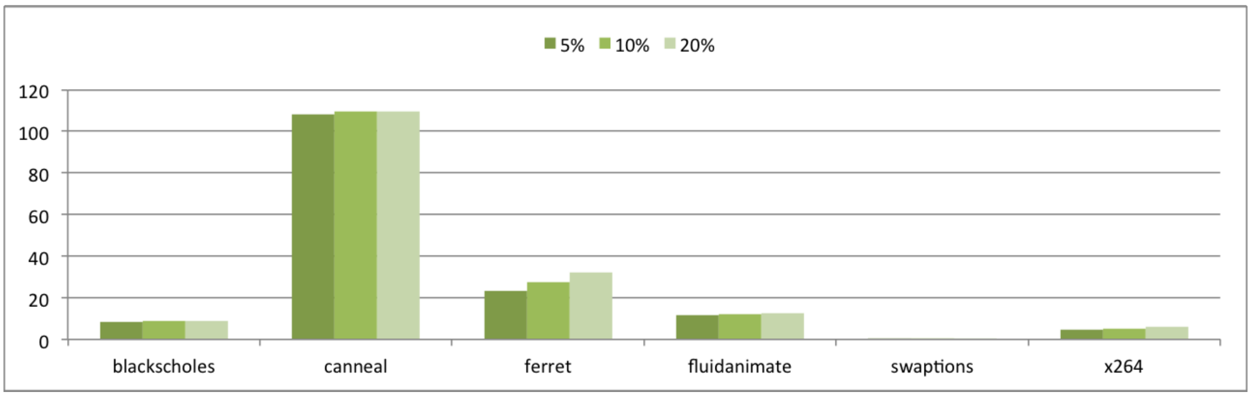
Fig. 1 (a-c) show final results of the experiments. Each figure includes the results for three levels of quality of service requirement, 5%, 10%, and 20%. In Fig. 1(a), the number of mispredictions per cache miss (MPCM) is shown. Mispredictions are the limiting factor in performance improvements achievable by ALVP as compared to LVP. ALVP pays a penalty for each misprediction in return for the ability to bound the error incurred by each approximation instance. This fact highlights the significant effect of tuning the relaxation window width on ALVP performance. An unnecessarily narrow relaxation window increases the rate of the cache misses that result in mispredictions. The prediction scheme as well as the mechanisms used for controlling the predictions (e.g. prediction confidence) can also play a significant role in determining MPCM.

Fig. 1(b) demonstrates how the MPCM metric translates into performance gains. Here, the number of recovered cycles per kilo-instructions (RCPKI) is shown. ALVP was able to improve the performance of `canneal` by as high as 107 cycles per kilo-instruction (more than 10 percent performance improvement) while delivering a guarantee of a less than 5% quality of service degradation. Except for `ferret`, increasing quality of service degradation bound has not resulted in noticeable performance gains. There are even cases such as `blackscholes` and `fluidanimate` in which the system has consumed part or all of its increased budget of inaccuracy (Fig. 1(c)) without providing

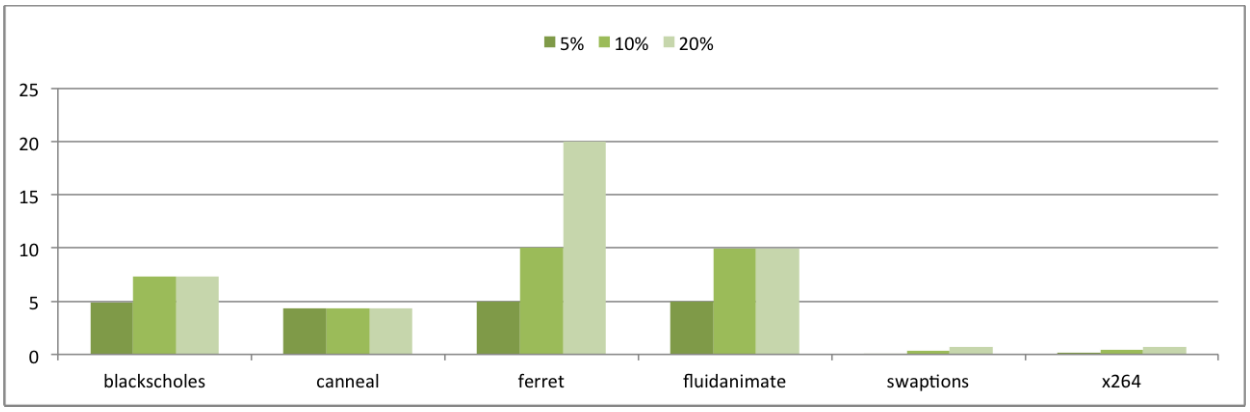
Figure 1: Experimental results with ALVP



(a) Misprediction per cache miss



(b) Recovered cycles per kilo-instructions



(c) Quality of service (error)

further speed-ups. This points to the need for improving the search algorithm so that it will not allow approximations that do not lead to performance benefits. This problem in particular seems to be rooted in the scoring metric used in the algorithm. A better scoring metric should sort the approximations so that the second phase does not run out of the inaccuracy budget before it has chance to include more approximations that could produce speed-ups.

3.3. Limitations

Full system simulation. We used estimates of total cache miss and value misprediction penalties to evaluate performance benefits of ALVP. More accurate performance evaluations need to perform full system simulations that consider out-of-order scheduling, contention and non-uniform cache miss and value misprediction penalties.

Static tuning. ALVP, as described in the previous section, tunes the width of relaxation windows for each instance of prediction in order to bound the approximation error. We chose to consider a static tuning approach instead. While this could limit the benefits of ALVP achieved by dynamic tuning, it enables the global optimization of the tuning configuration to get closer to the best possible performance. At the same time, although we implement a greedy search algorithm, its accuracy is still limited compared to an exhaustive algorithm.

Prediction scheme. We used a *last-outcome* prediction scheme which simply uses the result of the previous instance of the instruction as the prediction. Since the performance of ALVP is strongly dependent on the rate of mispredictions, a more sophisticated prediction scheme should be able to gain more benefits.

4. Related Work

Introducing approximation in load value prediction has only been investigated in two proposals. Rollback-Free Value Prediction (RFVP) [12] mainly focuses on GPUs and its main techniques are a subset of those evaluated in the other proposal, Load Value Approximation (LVA) [7]. Therefore, we focus on LVA as the closest related work.

In contrast to ALVP that introduces inexactness through relaxing the validation criteria, LVA eliminates the rollback mechanism altogether. While the error can still be measured after the missing data arrives at the processor, pre-approximation bounds cannot be placed on the approximation error. The complexity of the system is reduced while some of the ability to control the error is lost. To better compare and contrast LVA and ALVP, it is useful to think of LVA as two components: an approximate predictor and a prediction controller. The predictor approximates the result of selected load instructions and the controller monitors the accuracy of the approximations and decides when predictions are made and when the predictor is trained. For comparison to ALVP we are more concerned with the controller since both approaches can use the same predictor.

In LVA, the controller uses two parameters to control the predictor and navigate the tradeoff space:

Approximation degree specifies how many times an approximate value is re-used for prediction before the predictor is trained. An approximation degree of 4 for example means that a cache block fetch occurs once for every 5 cache misses. A higher approximation degree saves energy otherwise consumed to fetch from the cache but also increases the error since stale values could make predictions less accurate.

Relaxed confidence window defines how close the approximated value must be to the actual value to increment the confidence counter used to decide when predictions should be made. A larger confidence window increases the number of predictions but also decreases their accuracy and consequently the error.

Notice that while the approximation degree in ALVP is effectively zero, other values can be used, resulting in the same benefits and costs as in LVA. On the other hand, while the confidence window in LVA and the relaxation window in ALVP represent the same concept (defining what "close enough" means), they are used for different purposes. In ALVP, it is used to directly bound the approximation error by triggering the rollback mechanism in case of unacceptable errors. In LVA, this error control is implemented indirectly where the acceptable/unacceptable errors increase/decrease the predictor's confidence parameter, making future approximations more/less likely. Again, notice that these two mechanisms are not mutually exclusive and can be used at the same time. A major difference, however, is that the confidence window in LVA is a global value shared by all instructions and is set statically while the relaxation windows in ALVP are individually set for each instruction and can potentially change for each prediction instance.

5. Conclusions

We evaluated the potential performance benefits of Approximate Load Value Prediction (ALVP). First, we discussed the high costs associated with online monitoring and control of approximation in current software-based systems. We then explained how ALVP enables low-cost control of the error incurred by individual instances of approximation. Next, we presented the results of experiments that measured potential performance benefits of ALVP.

References

[1] Akram, A., and Sawalha, L. A comparison of x86 computer architecture simulators.

[2] Ansel, J., Chan, C., Wong, Y. L., Olszewski, M., Zhao, Q., Edelman, A., and Amarasinghe, S. Petabricks: A language and compiler for algorithmic choice. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2009), PLDI '09, ACM, pp. 38–49.

[3] Baek, W., and Chilimbi, T. M. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2010), PLDI '10, ACM, pp. 198–209.

[4] Esmaeilzadeh, H., Sampson, A., Ceze, L., and Burger, D. Neural acceleration for general-purpose approximate programs. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on* (Dec 2012), pp. 449–460.

[5] Kim, I., and Lipasti, M. H. Understanding scheduling replay schemes. In *In International Symposium on High-Performance Computer Architecture* (2004), pp. 198–209.

[6] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2005), PLDI '05, ACM, pp. 190–200.

[7] Miguel, J. S., Badr, M., and Jerger, N. E. Load value approximation. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2014), MICRO-47, IEEE Computer Society, pp. 127–139.

[8] Perais, A., and Seznec, A. Practical data value speculation for future high-end processors. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on* (Feb 2014), pp. 428–439.

[9] Samadi, M., Lee, J., Jamshidi, D. A., Hormati, A., and Mahlke, S. Sage: Self-tuning approximation for graphics engines. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (New York, NY, USA, 2013), MICRO-46, ACM, pp. 13–24.

[10] Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., and Grossman, D. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation* (New York, NY, USA, 2011), PLDI '11, ACM, pp. 164–174.

[11] Sidiroglou-Douskos, S., Misailovic, S., Hoffmann, H., and Rinard, M. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (New York, NY, USA, 2011), ESEC/FSE '11, ACM, pp. 124–134.

[12] Thwaites, B., Pekhimenko, G., Esmaeilzadeh, H., Yazdanbakhsh, A., Mutlu, O., Park, J., Mururu, G., and Mowry, T. Rollback-free value prediction with approximate loads. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation* (New York, NY, USA, 2014), PACT '14, ACM, pp. 493–494.

[13] Zhang, Q., Yuan, F., Ye, R., and Xu, Q. Approxit: An approximate computing framework for iterative methods. In *Proceedings of the 51st Annual Design Automation Conference* (New York, NY, USA, 2014), DAC '14, ACM, pp. 97:1–97:6.