Preliminary Experiments on Similar Executions with Reduced Off-Chip Accesses in Multi-core Processors

Susmit Biswas, Frederic T. Chong, Diana Franklin, Timothy Sherwood
{susmit,chong,franklin,sherwood}@cs.ucsb.edu
University of California, Santa Barbara, CA - 93106, USA

Abstract

With the increasing number of cores in Multicore processors, limitations in memory bandwidth are a significant issue. We find that there exists a high degree of similarity across multiple executions of the same application with minor variations in input parameter values or input data sets. In this work, we examine two applications where individual executions on different cores work with different parameters or input data in practical scenarios, and we show that a large fraction of the cached data is common across cores. We propose a *Merge-Cache*, which requires minor architectural modifications to leverage this phenomenon by merging cache lines owned by different processors, which contain identical data. By merging identical cache lines, effective cache capacity per process increases, leading to a reduction in off-chip memory accesses. In this paper, we present initial experimental results for two benchmarks *ammp* and *twolf* from SPEC2000-CPU suite, and show that our proposed technique reduces off-chip memory access by a factor of three on average when eight identical instances of the benchmark executes in parallel with minor modification in parameter values.

1 Introduction

The increasing gap between memory and processor speed has posed a significant challenge in memory system design. While individual processor performance is no longer increasing with Moore's law, the demand for "More than Moore" has motivated researchers to explore innovations in multiprocessor domain. However, the problem of limited memory bandwidth remains unsolved. Past work has tried to mitigate bandwidth issues through efficient data partitioning[5], cooperative caching[4], streaming cache from one processor to another etc. It will be, however, difficult to sustain bandwidth demands when number of cores in a chip scales to tens or even hundreds.

In this paper, we show that most of the accesses to memory operate on the same data when the same application is executed in multiple processors in parallel with different input data. Therefore, multicore processors can be used more effectively in these scenarios by merging identical cache lines. We propose the *Merge-Cache* architecture to exploit this phenomenon by providing support for merging cache lines. There are several practical scenarios in domains of simulation, visualization, security etc. where multiple instances of same application are executed with minor variation of parameters or input data *e.g.* the device fabrication process requires many Monte Carlo simulations[8] with minor variations in device parameters to design variation-tolerant devices. In the machine learning domain, ensemble learning[10] techniques use several poor learners to develop finer models. We expect our technique to improve the performance of these applications. We have implemented a trace-based simulation framework to demonstrate the strength of our approach. In this paper, we present results for two applications, *ammp* and *twolf* from SPEC2000 benchmark suite[1], and show that merging cache lines with similar data reduces in memory accesses

by an order of magnitude.

The remainder of the paper is organized as follows. In section 2, we describe previous approaches to reduce memory accesses and explain our technique in section 3. We illustrate the experimental methodology in section 4, present results in section 5. We discuss the results in in section 6 with a perspective on future work in section 7.

2 Related Work

Several prior proposals use compiler and architectural support to reduce main memory access and in turn speed up execution. In order to reduce memory stalls, Mahlke et al.[5] propose a profile-guided data partitioning technique. Thread level speculation[3][12] using compiler and architectural support speeds up application execution by spawning speculative threads. Though, these techniques speed up execution significantly, with increasing number of cores in a chip, the demand for memory bandwidth is also increased.

In order to reduce memory access, several cache optimization schemes have been proposed. Chang et al. proposed cooperative caching technique[4] in a multiprocessor to reduce off-chip access using a cooperative private cache either by storing single copy of clean blocks or providing a victim cache like spill-over memory for storing evicted cache lines. An orthogonal study, which has similar motivation as our work, is the data cache compression technique as proposed by Almadeen et al [2]. Compressing the L2 data results in reduction in the cache space required to store data. The authors reduce the off-chip accesses and thus save bandwidth.

Another technique which motivates our approach is *copy-on-write* mechanism used in virtual machines and operating systems. In copy-on-write technique data initially shared by multiple processes become different once one of them write to it and separated memory regions never merge again. In our scheme, cache lines are merged at memory write operation, and sharing is done at finer granularity

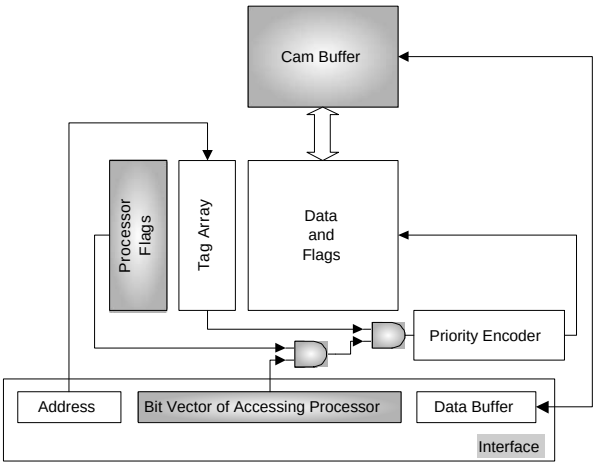


Figure 1: Proposed cache architecture. An array of flags (*processor-flags*) indicates presence of a block for different processors. For *read* operations, processor flag is also checked corresponding to the requesting processor. At *write* operations, lines having same address as the evicted line are brought to CAM and searched for identical content.

too in order to support generalized classes of applications. Sorin et al. proposed Multiversion Memory[11] to increases fault tolerance by storing versions of the data. We take a different approach in this work and propose merging similar data to reduce main memory access.

3 Architectural Support for Merging Cachelines

In conventional caching techniques, data search is guided by its address. Searching for identical data from different processes, however, can be extremely expensive unless the search space is restricted. We notice that data similarity can be discovered by virtual address of the process as data allocated at the same virtual address usually match across processes. Lower level caches are typically indexed by physical address instead of virtual address. Therefore, searching for identical data and employing an efficient organization is a challenge.

In this paper, we propose minor modifications over conventional cache architecture for supporting dynamic data merging, as shown in

Figure 1. We show experimental results only for L2 cache though our technique can be employed in all lower level caches.

Every line in the cache is augmented with a bit vector (*processor flags*) of length equal to the number of processors and virtual address is used for tagging. Virtual address aliasing problem is addressed using a RTLB. Whenever a *read/write* operation is requested by a processor, the corresponding *processor flag* is also checked, and an access is considered a *hit* only if the tag matches and the flag corresponding to the requesting processor is set too. A *bitwise-and* operation with the requesting processor bit chooses the correct cache line. As the processor flag matching can be performed in parallel to tag matching, it does not add delay in critical path. The only added delay corresponds to two added AND operations as shown in Figure 1. For a write operation, data values from lines having identical address as the referenced line are copied to the associative buffer (CAM), which is then searched for the content of the line being written to L2.

As the rate of writing to L2 is less than the L1 miss rate, this operation does not add significant overhead in L2 access. The overhead of incorporating the additional associative buffer or CAM, extracted using Cacti 4.2[9], is enumerated in Table 1. Note that the overhead of using a 16 entry CAM is negligible when compared to the entire cache. A typical 4 MB, 16-way L2 cache, partitioned in 8 banks, consumes $16.39mm^2$ of area, $3.74W$ dynamic read power and $3.93ms$ access time in $45nm$ technology node (computed using Cacti 4.2). The cache management logic is also modified for searching lines containing the same data at write operations, as shown in Figure 3. Note that, exclusive L2 cache benefits more as communication between L1 and L2 cache is minimized and total on-chip cache also is increased.

4 Methodology

In this section, we describe the evaluation framework and benchmark applications. We have implemented a trace-based simulation framework to evaluate the effectiveness of our scheme. The simulation infrastructure consists of two components:

1. A Pin-based memory trace generator which monitors memory references [7], and
2. A memory-access analyzer which implements a trace-based multiprocessor cache simulator.

We implemented the following three cache architectures for an exclusive L2 cache. (a) Every processor has a private L2 cache along with private L1. (b) A large L2 cache is shared by all the processors and the size of the cache equals the cumulative amount of L2 cache in private cache architecture. (c) A shared L2 Merge-cache where cache lines are merged if cache contents match.

In this work, we perform sensitivity analysis of cache line merging efficiency on cache size, associativity and number of processors sharing a cache. We assume that only a core is allocated only to a single process.

In order to analyze the dependence on cache size, we simulate a scenario with fixed number of processes (8) sharing a 8-way set associative cache while varying cache size from 512 KB to 128 MB. In case of the private L2 cache architecture, the cache is physically distributed among all the processors.

For sensitivity analysis on associativity of the cache, we simulate 8 processes with 4 MB cumulative cache size while varying associativity from 2 to 16. We also vary number of processes/processors sharing a cache, in another set of experiments with a 4 MB, 8-way set associative cache, to capture the scalability of Merge-cache architecture. In this set of experiments, we choose a subset of inputs from a larger set of inputs for scenarios with smaller number of processes. In all experiments, parameters are varied randomly around a mean value with maximum variation of 50%.

In the remainder of this section, we describe the benchmarks and the parameters that we vary in our experiments. We have selected two

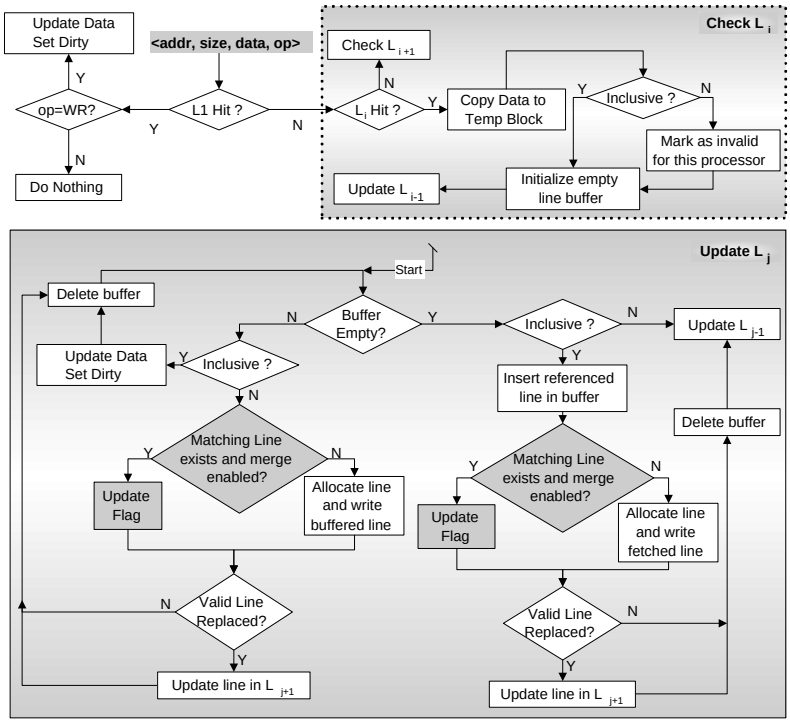


Figure 2: Modifications to cache management technique and cache architecture are indicated as shaded blocks. When a line is evicted from L1 cache, it is merged with pre-existing cache line in L2 if their contents match. We implement exclusive L2 cache as it increases total on-chip memory.

Rows	Area (mm^2)	Access time (ns)	Power (nW)
2	0.0132	0.507	0.0048
4	0.0140	0.513	0.0055
8	0.0156	0.525	0.0071
16	0.0190	0.549	0.0102

Table 1: Overhead of 256-bit wide CAM obtained using Cacti 4.2 for $45nm$ technology node. Required number of rows in the CAM equals the associativity of Merge-cache.

applications, *ammp* from SPEC CFP2000 and *twolf* from SPEC CINT2000 to show that data similarity is observed in both classes of applications.

4.1 ammp

ammp, which belongs to SPEC CFP2000 benchmark suite[1], solves a problem in computational chemistry using molecular dynamics where energy of the final configuration of a set of atoms in water and protein is computed. We have chosen the following parameters which could be varied by researchers in practical experiments.

- *mmbox* controls fast multipole algorithm (FMM) for long-range non-bond energy calculation, and when set to a non-zero functions as a factor to compromise between accuracy and speed.
- *bbox* is the bounding box dimension used for computing potential energy.
- *mxdq* works as a threshold to update the full non-bonded list when atomic dis-

placement is greater than the value in angstroms.

- *temp* specifies the simulation temperature. Simulation can be performed with different values to measure sensitivity to temperature.
- *numstep* specifies the number of steps used in the line minimizer.

The *ref* inputs distributed with SPEC benchmark suite are very large for detailed simulations whereas Minnespec[6] inputs are small for simulating large cache behavior due to warm-up time. Therefore, we have modified the input set of ammp following the Minnespec[6] guidelines such that the average virtual memory and resident memory size are 15.71 MB and 14.38 MB respectively, and the number of references simulated are around 2.21 billion.

4.2 Twolf

The TimberWolfSC placement and global routing package is used in the process of creating the lithography artwork needed for the production of microchips in practice. TimberWolfSC program uses simulated annealing as a heuristic to find good solutions for row-based standard cell design style. The global router requires to add extra cells known as *feedthrus* to complete the route if not enough space is present between two adjacent standard cells. A valid placement is one in which all of the cells are placed within the specified rows without any overlap between cells. We vary the following parameters to find optimal routing. In practice, many simulations need to be run in order to discover the "magic numbers" for optimal routing.

- *rowSep* is the gap between two rows.
- *feedthruwidth* is width of extra cell used to assist in routing completion.

The inputs to this benchmark are modified so that the average virtual memory and resident memory size for twolf are 2.67 MB and 1.35 MB respectively when number of references simulated are approximately 4.13 Billion.

5 Results

In this section, we present our observations on data similarity between executions of the same application with different input conditions, and show the reduction in memory access by using cache line merging technique.

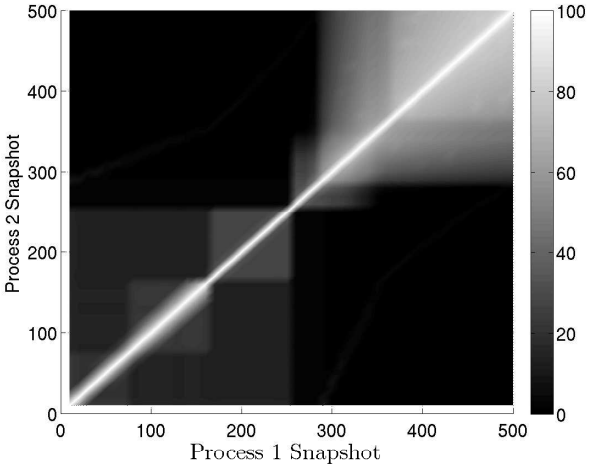
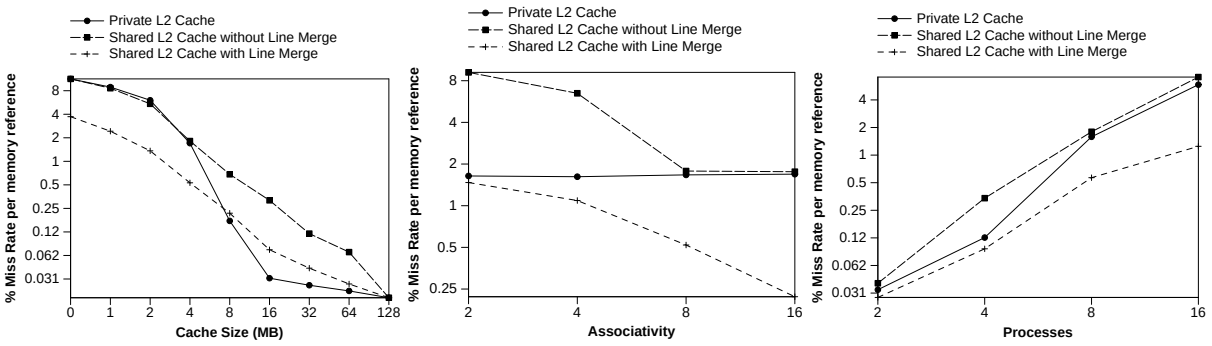


Figure 3: Similarity of cache contents between two instances of *ammp* running with different sets of parameters. The simulations were performed with 1MB direct-mapped cache for first 500 Million references in each execution, which show very high similarity across the diagonal. As we observe regions with high similarity in close vicinity of the diagonal, it can be inferred that the executions own quite similar working sets even if they are not perfectly synchronized.

In Figure 3, we show the similarity of the cache contents while simulating *ammp* on the same input data, but with variations up to 50% in parameter values. Cache snapshots are taken every 10M accesses, and we present the result for the first 500 Million accesses only here, but the same behavior is observed throughout the execution. We observe that similarity across executions can be exploited in *ammp* if the executions start at the same time and are synchronized with each other. Note that the resident memory size of *ammp* is larger than 1 MB though the cache contents are very



(a) Dependence on cache size

(b) Dependence on cache associativity

(c) Scalability of Merge-cache

Figure 4: **ammp**:L2 miss rate per memory reference with various numbers of executions, cache size and associativity. In case of private L2 cache, the total cache is partitioned equally among n threads individually. Data merging technique decreases L2 miss rate by an order of magnitude from traditional shared and private cache. Note that logarithmic scale has been used in vertical axis for all these results.

similar. We leverage this similarity using our cache line merging technique and present results in the rest of this section.

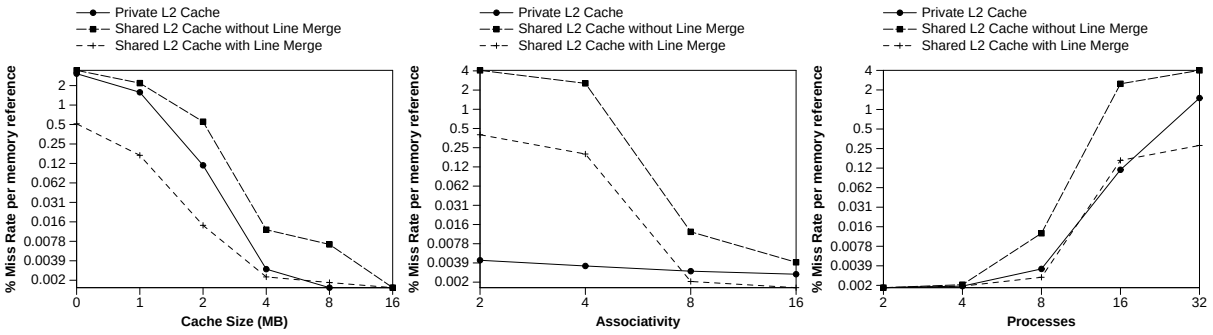
In Figure 4 and Figure 5, we show that the L2 cache miss rate is reduced significantly by merging cache lines. It can be observed from Figure 4(a) and Figure 5(a) that merging cache line results in a reduction of main memory access by an order of magnitude if the cache is not able to retain the whole memory footprint. When the cache is large enough to store the entire working set, all the cache architectures perform equally well. One interesting behavior that we can note from these graphs is that same cache performance is achieved with a smaller sized Merge-cache. With scaling of number of cores in a processor, per core effective cache size decreases magnifying cache miss rate, which can be improved using a Merge-cache architecture. Note that the results are shown on a logarithmic scale.

As depicted in Figure 5(b) and 4(b), a shared cache with low associativity shows worse cache behavior than a private cache with equal associativity. In a shared cache, the number of sets in the cache increase, but associativity stays same, leading to degradation in cache performance. Note that a cache employing merging scheme shows better cache performance due to increased cache capacity due to line merging.

Finally, we present our results on scalability aspect of cache line merging technique. Due to constraints in resources, we simulate up to 32 simultaneously executing processes and present the results. As the number of cores in a processor scale, effective per core cache capacity decreases leading to an increase in cache miss rate, which can be observed clearly in Figure 4(c) and Figure 5(c). Merging identical cache lines increases per core effective cache capacity, and thereby, enhances cache performance. With scaling in number of cores, this effect is more pronounced as the technique of merging cache lines reduces cache miss rate by orders of magnitude and accesses to off-chip memory in turn. Specifically, for 8 simultaneously executing instances of *ammp*, we observe reduction in cache miss rate by 3×. With scaling of number of cores per chip, we expect to observe more benefit of employing line merging technique in cache for similar executions.

6 Conclusion

In this paper, we have shown that a large amount of data is identical between individual



(a) Dependence on cache size

(b) Dependence on cache associativity

(c) Scalability of Merge-cache

Figure 5: **twolf**:L2 miss rate per memory reference with various numbers of executions, cache size and associativity. In case of private L2 cache, the total cache is partitioned into n parts and used by n threads individually. Data-merging technique decreases L2 miss rate by an order of magnitude if associativity is not very low, as having shared L2 results in increased conflicts. We address this issue in section 6. It can be observed that same cache performance is achievable using a smaller Merge-cache.

executions of the same application though they work with different parameters. Executing multiple instances of an application with similar data sets or parameters is observed in many practical scenarios. We present our preliminary study on two benchmarks *ammp* and *twolf* from SPEC2000 benchmark suite. We propose Merge-cache architecture which requires minor modifications in conventional cache systems to leverage the similarity in cache contents of these applications by merging identical cache lines. Merge cache increases per core effective cache capacity by compacting cached data, which in turn reduces main memory access by an order of magnitude.

7 Future Work

Initial results on two SPEC2000 applications, *ammp* and *twolf*, demonstrate the potential of this approach. In shared L2 cache setting, all processors use the same function for mapping addresses to cache sets, which leads to increased conflicts due to dissimilarity in some cache lines. In traditional L2 cache, this issue is addressed by using a different mapping function for every processor, which makes searching for mergeable data difficult. For cases with high similarity in execution, this problem is avoided because of the tremendous amount of merging of data, but we plan to address it by using a split cache which we have left as future work.

In this paper we have explored only the data similarity across many executions of an application. The benefit, however, could be larger due to instruction cache similarity too, which we aim to evaluate in future. Our approach has shown encouraging results for applications from several other domains such as visualization, machine learning too. We plan to explore these applications as part of our future work.

8 Acknowledgements

This work was supported in part by NSF award 0627749, an AFOSR MURI to the Helix project, by NSF MRI-061991, and by CAREER Grant award NSF-0619911 to Diana Franklin. We would also like to thank Ayswarya Sundarum, Alan Savage, Ryan Dixon, and Vlasia Agnostopoulos for their con-

tributions to this project.

References

[1] SPEC CPU2000: http://www.spec.org/cpu/.

[2] A. Alameldeen and D. Wood. Adaptive Cache Compression for High-Performance Processors. In *31 st Annual International Symposium on Computer Architecture*, June 2004.

[3] M. J. Bridges, N. Vachharajani, Y. Zhang, T. Jablin, and D. I. August. Revisiting the Sequential Programming Model for Multi-Core. In *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, December 2007.

[4] J. Chang and G. S. Sohi. Cooperative Caching for Chip Multiprocessors. In *ISCA '06: Proceedings of the 33rd Annual International Symposium on Computer Architecture*, pages 264–276, Washington, DC, USA, 2006. IEEE Computer Society.

[5] M. Chu, R. Ravindran, and S. Mahlke. Data Access Partitioning for Fine-grain Parallelism on Multicore Architectures. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 369–380, Washington, DC, USA, 2007. IEEE Computer Society.

[6] A. J. KleinOsowski and D. J. Lilja. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research. *IEEE Computer Architecture Letters*, 1(1):7, 2006.

[7] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building Customized Program Analysis Tools With Dynamic Instrumentation. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–200, New York, NY, USA, 2005. ACM.

[8] M. Nedjalkov, H. Kosina, and S. Selberherr. Monte Carlo Algorithms for Stationary Device Simulations. *Mathematics and Computers in Simulation*, 62(3-6):453–461, 2003.

[9] S. Wilton and N. Jouppi. CACTI: An Enhanced Cache Access and Cycle Time Model. *IEEE Journal of Solid-State Circuits*, 31(5):677–688, May 1996.

[10] P. Sollich and A. Krogh. Learning With Ensembles: How Overfitting Can Be Useful. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 190–196. The MIT Press, 1996.

[11] D. J. Sorin, M. M. K. Martin, M. D. Hill, and D. A. Wood. Fast Checkpoint/Recovery to Support Kilo-Instruction Speculation and Hardware Fault Tolerance. (TR-1420), October 2000.

[12] J. G. Steffan, C. Colohan, A. Zhai, and T. C. Mowry. The STAMPede Approach to Thread-Level Speculation. *ACM Transactions on Computer Systems*, 23(3):253–300, 2005.