1 Summary

Designing and Evaluation of Compiler Algorithm for Prefetching (1992):

2 Strengths

- 633 citations

3 Weaknesses

- Section 2 where they discuss their prefetch algorithm seems more verbose than it needed to be; however, this is coming from the perspective of someone reading their study 30 years later and having learnt about the concept of prefetching and loop unrolling already. The entire section references the same Figure 2 throughout, but it would have been better done both aggregated and broken up, where each new term and strategy they introduced was accompanied by a visual example of how the locality, loop peeling or runrolling, and/or how the prefetching predicate worked with the example.
- 633 citations; they do software prefetching which inserts instructions; papers as recently as 2022 still Mowry (1992), but more an more, this seems to be in the context of background information for their papers. This is Mowry's PhD thesis. Other similar papers in the early 90s also covered software controlled prefetching.
- The scope of the compiler algorithm was limited to affine array accesses within scientific applications. Could they have also explored other access patterns in this study (mentioned in sec 6)? If not, it makes sense becauase they did things like assumptions about prefetch accesses not being able to be interrupted, for some of the benchmarks, they manually changed the alignments of the matrices to reduce cache conflicts

4 Rating: 4

5 Comments

6 Notes

- Problems in software-controlled prefetching: incurs instruction overhead, increas load on memory subsystem.
- Address: reduce prefetches (overhead) for data already in cache.
- Their algorithm: identifies memory references that are likely to be cache misses and only issues prefetch instructions for those.

  - Greatly improves performance
  - Better than a prefetch algo that does so indiscriminitely.

- They assert that future microprocessors should support memory hierarchy optimizations, i.e. lock-up free caches (allow multiple outstanding misses; ie in prefetch, the software uses instructions to pre-fetch/get data before the normal fetch, which is a "miss" om a sense) and ISAs should have prefetch instructions. (complex hardware prefetching is not necessary).

- **Locality Analysis**:

  - intrinsit data reuse (in a loop nest): find instances of data (array) accesses that refer to the same cache line. Temporal, spatial, and group reuses.
  - exploit set of reuses for cache of particular size

- They demonstrate the benefits of software controlled prefetching, ie speedup in overall performance, with typically small prefetching memory overhead and sometimes the number of instrs actually decreasings due to savings through loop unrolling.

- I think Fig 3, 4 is good representaation of results.

- Aspects of Prefetching alforithm:

  - Locality analysis - eliminate prefetching overhead by only prefetching for cache miss references (selective prefetching, rather than indiscriminate). Fig. 4 compares performance for no pf, indiscriminate pf, and selective pf. indiscriminate pf suffers from increased instr overhead and stress on memory subsystem, whilc selective pf improve performance and even resulted in gains in some benchmarks.
  - Loop splitting: selective pf w/ conditional statements vs. selective pf w/loop splitting. the latter performed better, evidence by the instr overhead per prefetch issue.
  - Software Pipelining:

References

[1] Todd C. Mowry, Monica S. Lam, and Anoop Gupta. 1992. Design and evaluation of a compiler algorithm for prefetching. In Proceedings of the fifth international conference on Architectural support for programming languages and operating systems (ASPLOS V), Richard L. Wexelblat (Ed.). ACM, New York, NY, USA, 62-73. DOI=http://dx.doi.org/10.1145/143365.143488