Selective Fill Data Cache

Final Presentation

Anuj Dharia
Paul Rodriguez
Ryan Verret

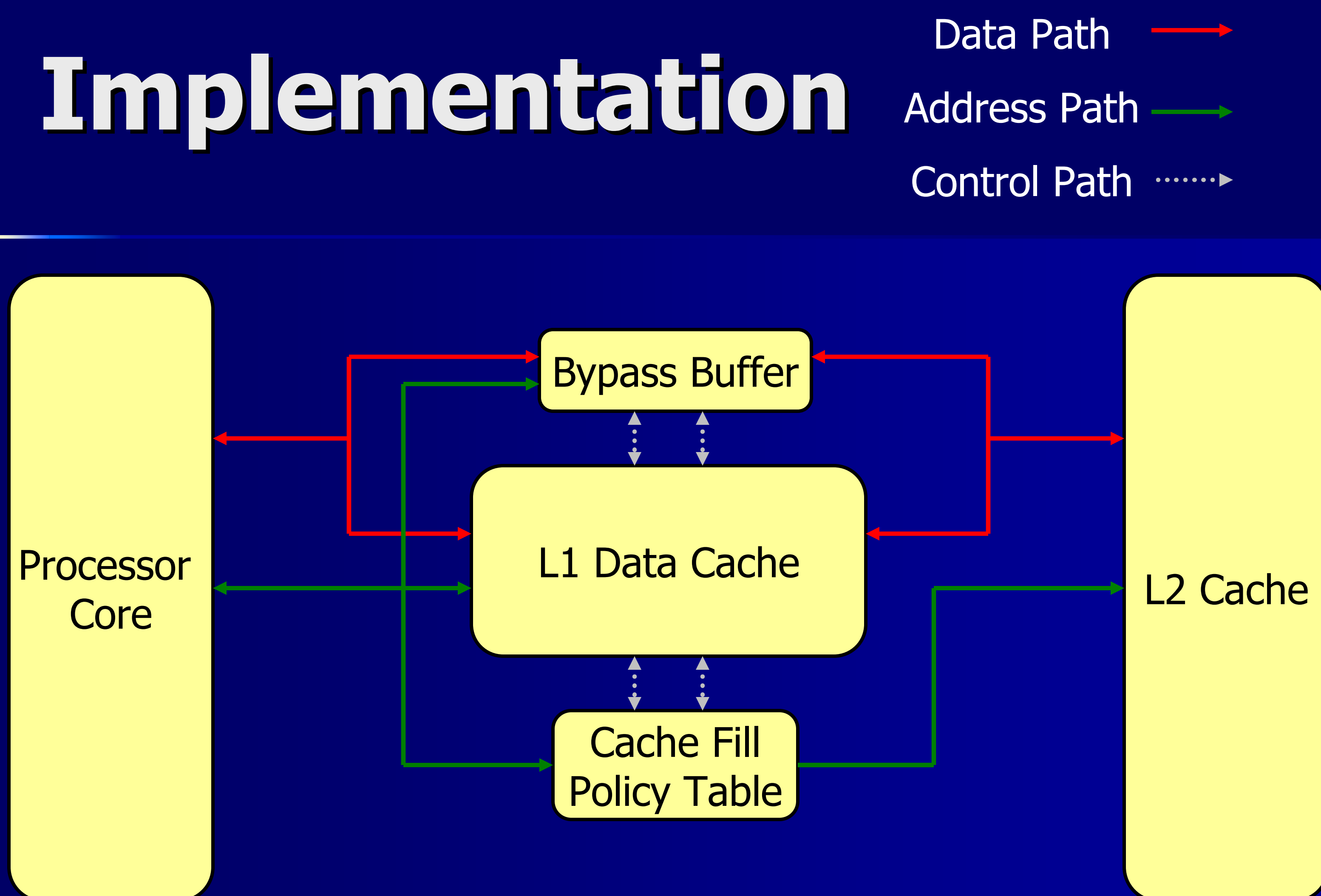
Motivation

- Superscalar clock frequency and issue width continue to increase
- Cache sizes remain the same or decrease to accommodate the clock
- More frequent data accesses place greater strain on cache
- Must be more selective about what is allowed in cache

Hypothesis

Cache efficiency can be improved by selectively preventing infrequently used data blocks from filling the L1 cache. Data consistently evicted from the cache before a subsequent access ought not enter if it is to evict useful data.

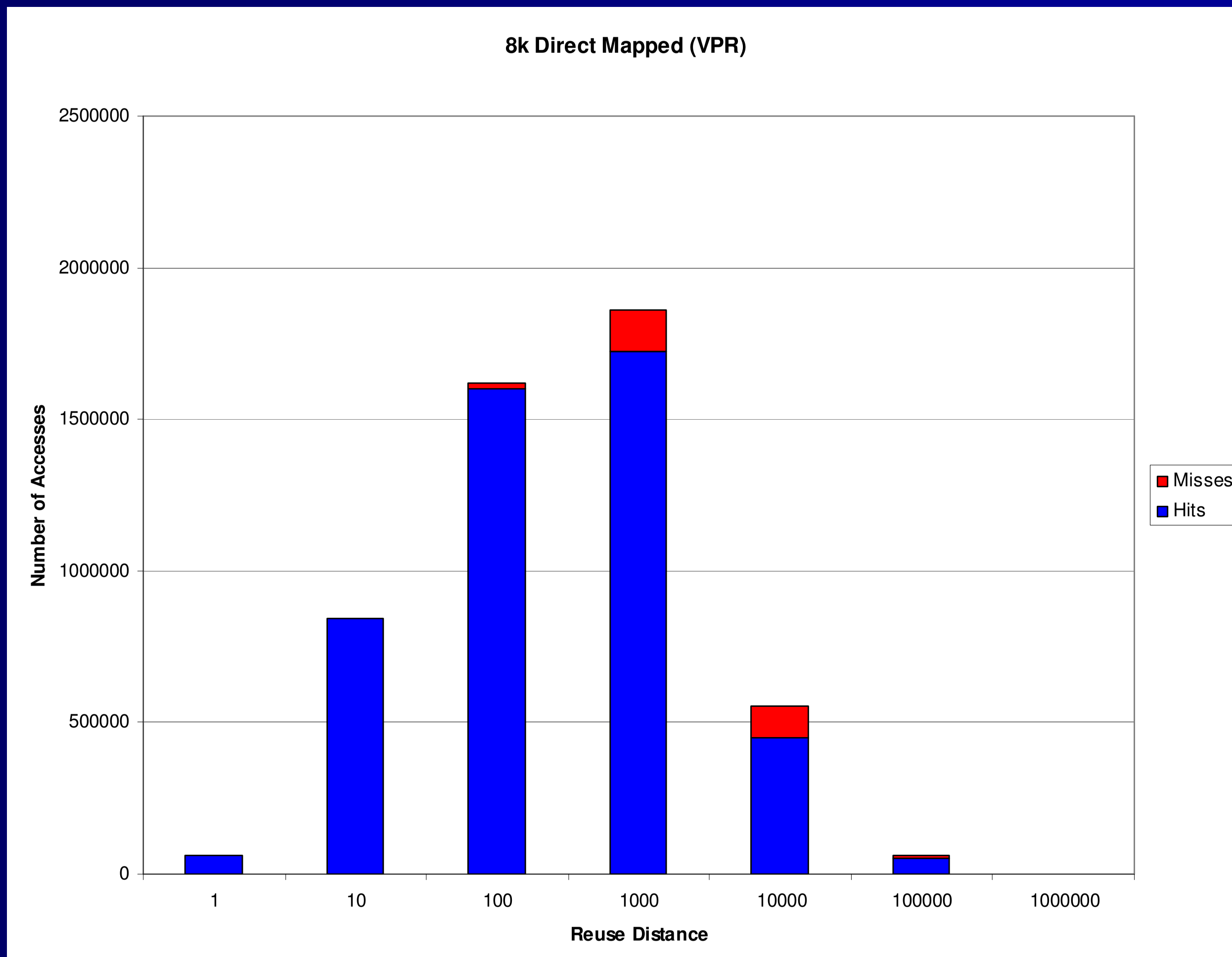
Implementation



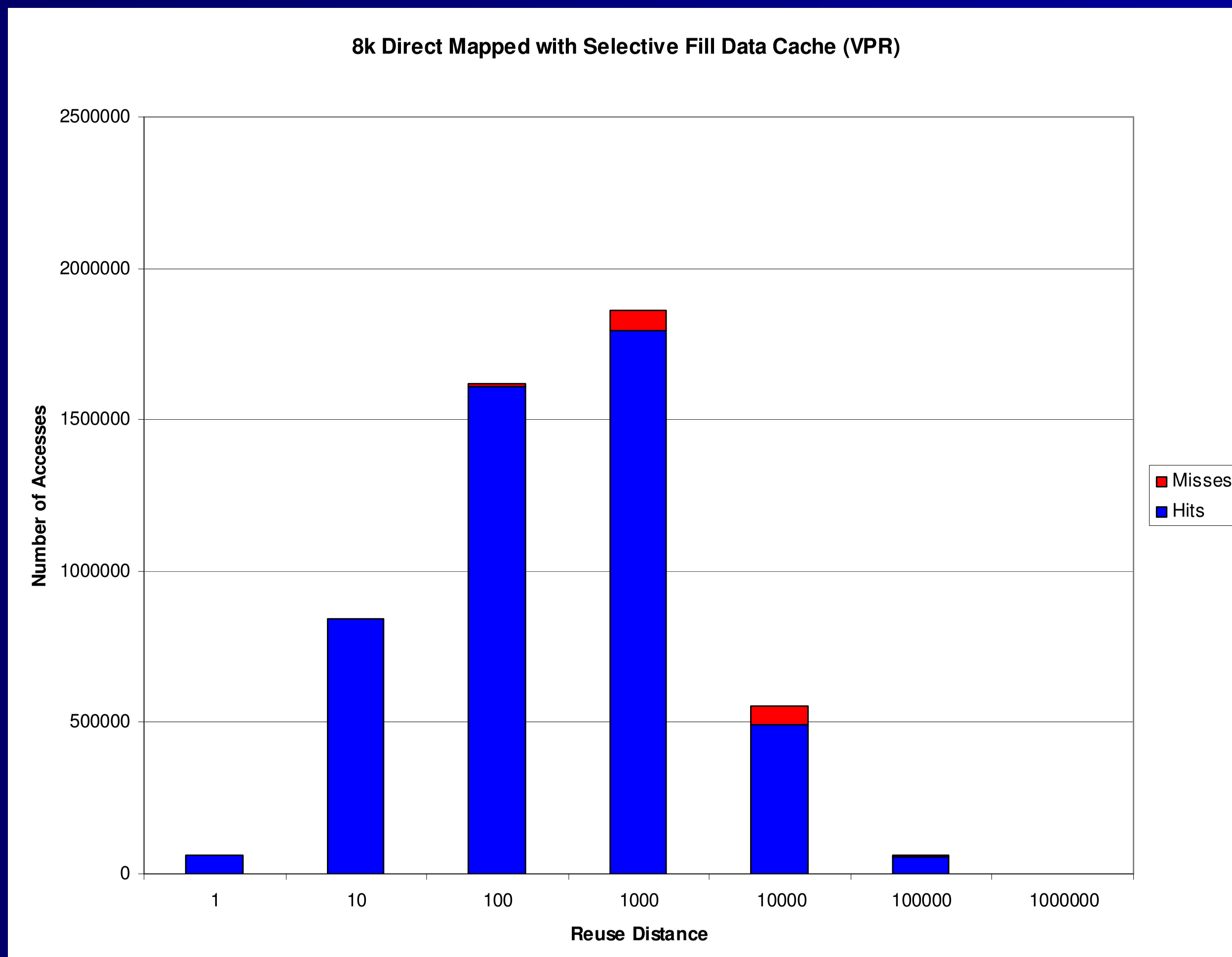
Architecture Details

- L1 Data Cache
  - Used bits
- Cache Fill Policy Table
  - Direct mapped
  - Size proportional to number of sets
  - Contains threshold information
- Bypass Buffer
  - 2-way set associative
  - Variable size (optimally 1/16 the total cache size)

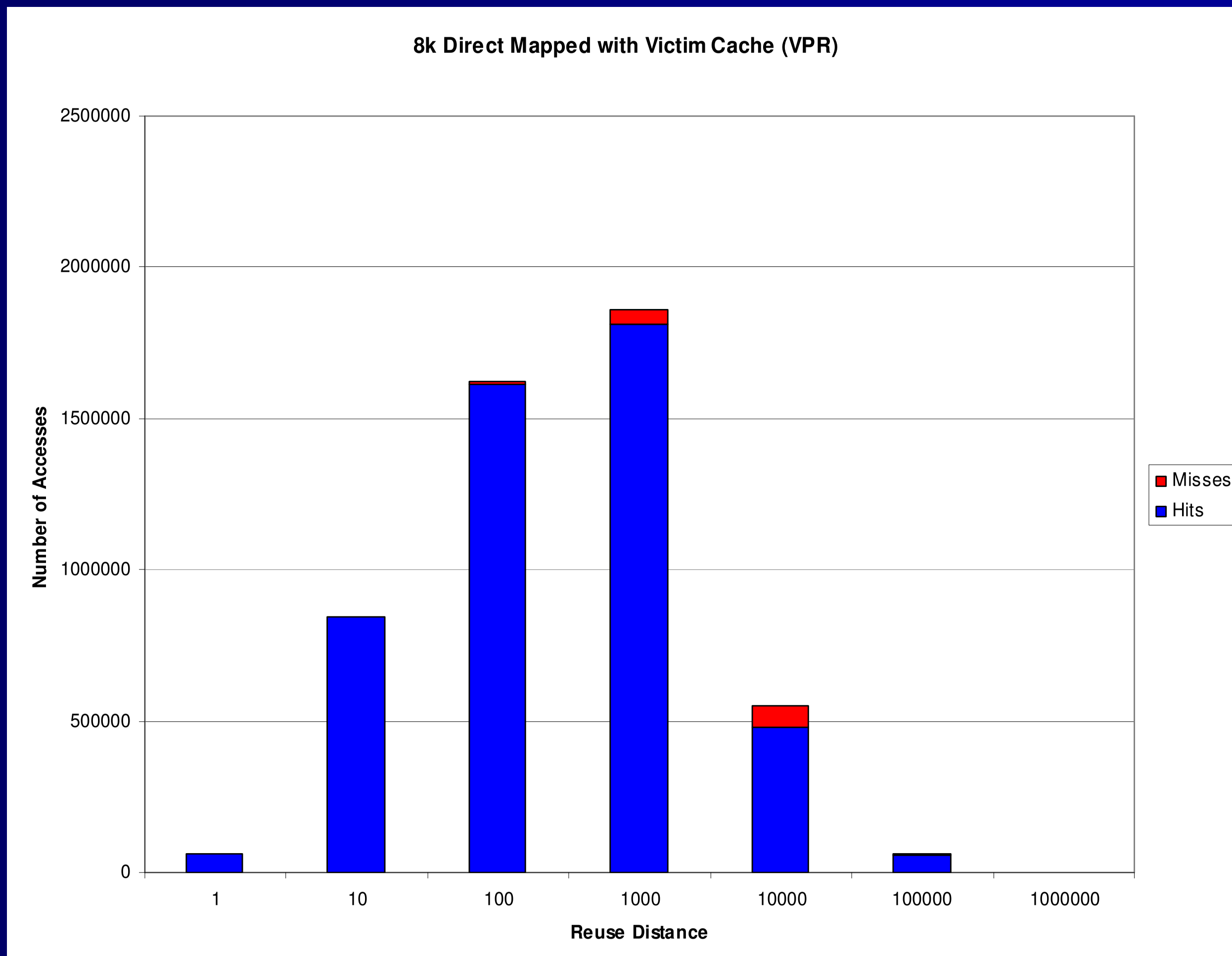
Base Configuration



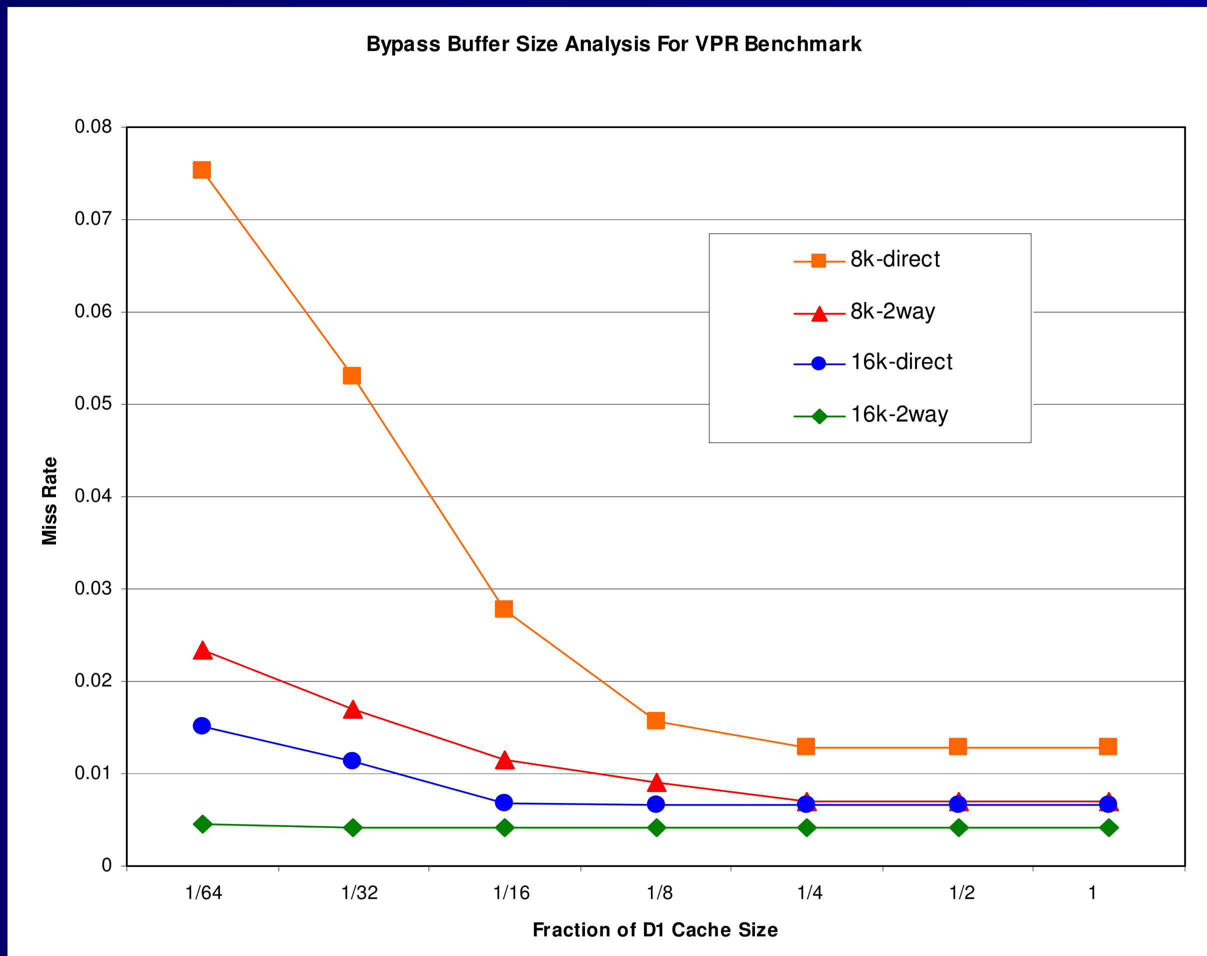
Selective Fill Data Cache



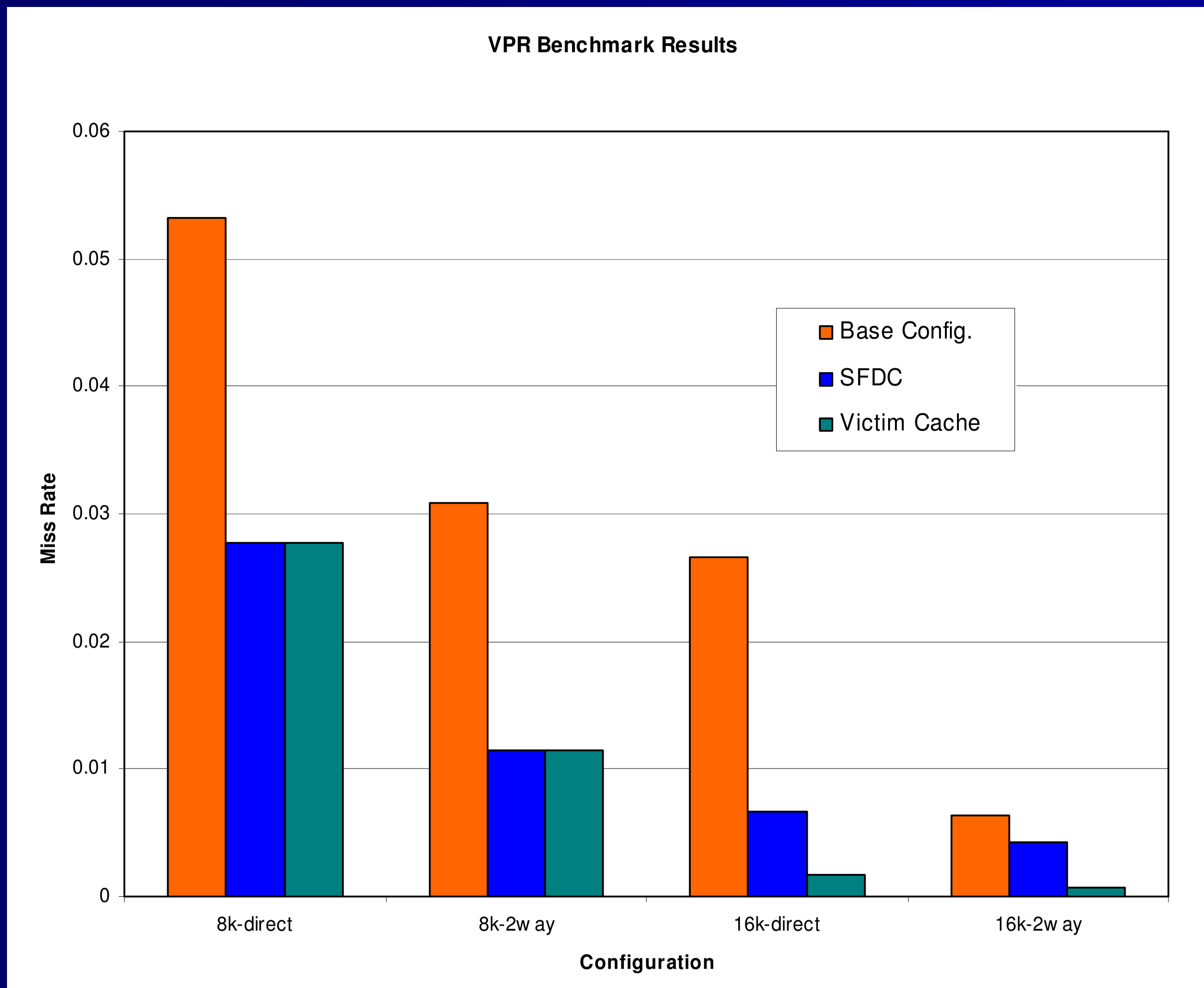
Victim Cache



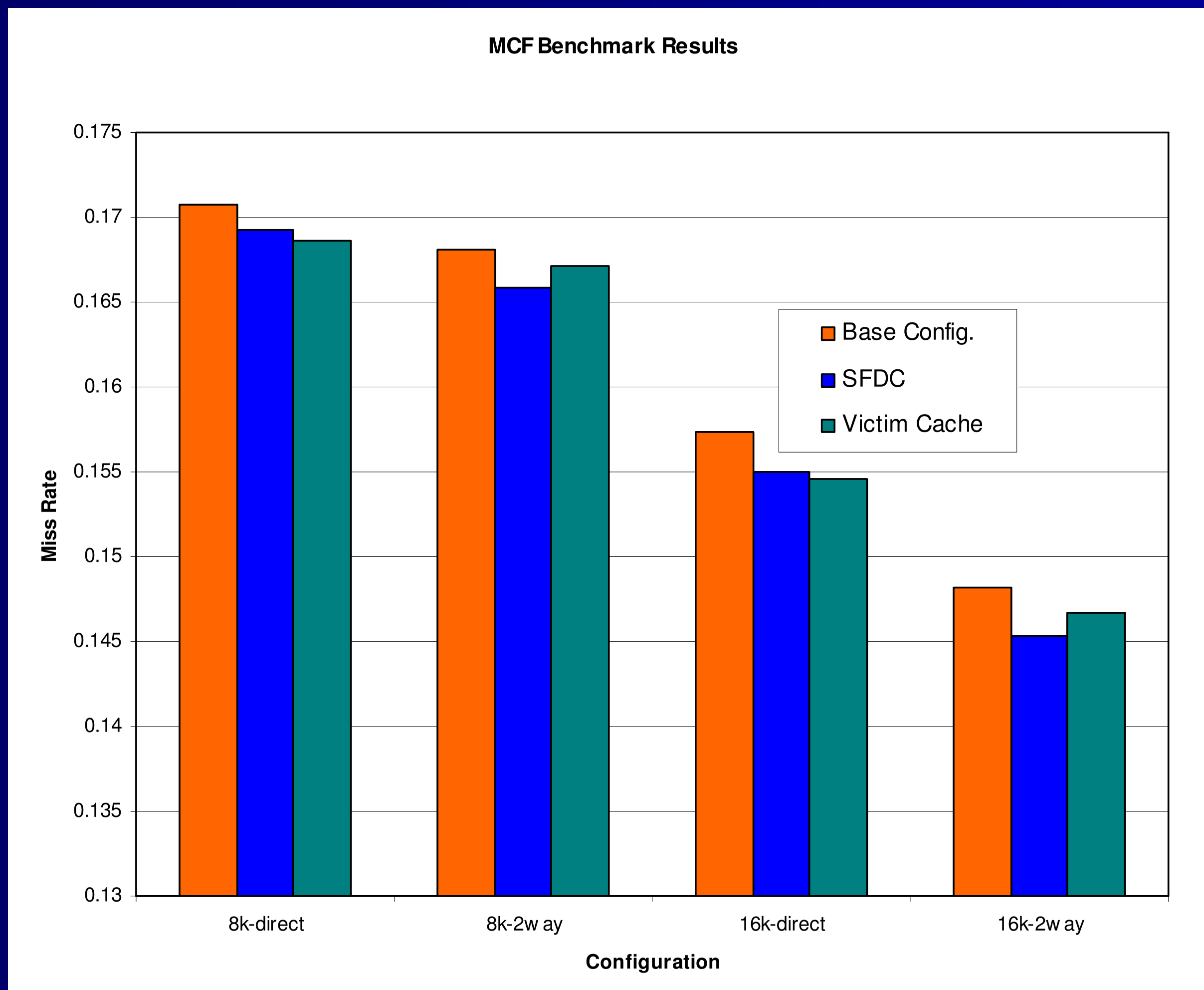
Bypass Buffer Analysis



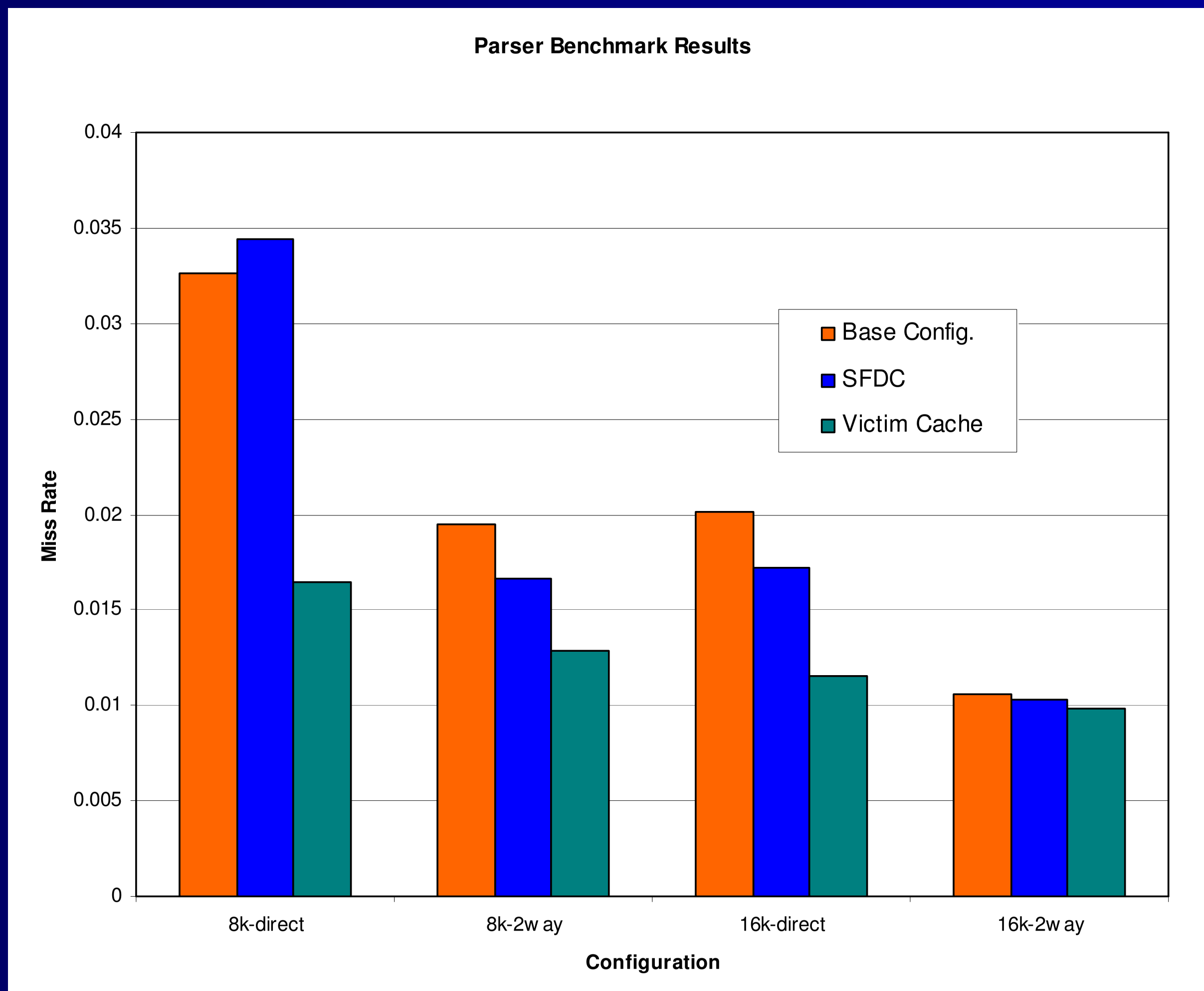
VPR Benchmark



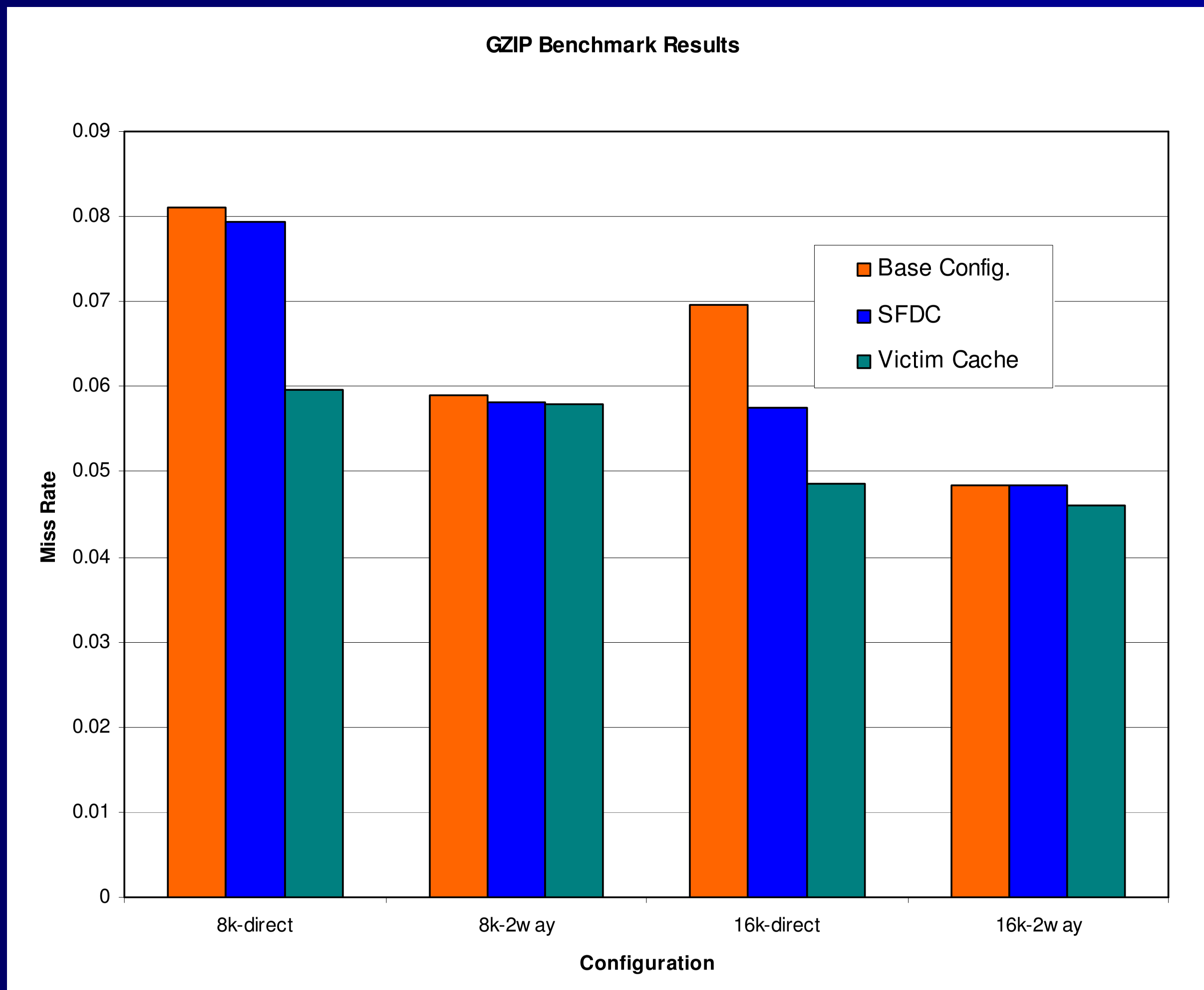
MCF Benchmark



Parser Benchmark



GZIP Benchmark



Future Work

- Implement both SFDC and victim cache together to see if improvement is complimentary
- Reevaluate the tradeoffs between SFDC and victim cache to take access time into account
- Eviction from cache fill policy table
- Dynamically determining optimal thresholds

Hypothesis Revisited

Cache efficiency can be improved for most benchmarks by selectively preventing infrequently used data blocks from filling the L1 cache. Data consistently evicted from the cache before a subsequent access ought not enter if it is to evict useful data.

Conclusions

- SFDC successfully improves cache performance
- In terms of area only, a victim cache outperforms an SFDC
- SFDC works better than a victim cache for larger reuse distances
- Temporal locality can be better exploited with a more advanced cache architecture

Questions?