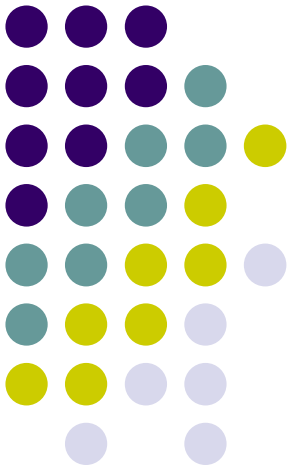
# Elec525

Project:

Tradeoff between accuracy of a modified Markov prefetcher to memory bandwidth usage and overall execution performance

April 23, 2005

Raj Bandyopadhyay, Mandy Liu, Nico Peña

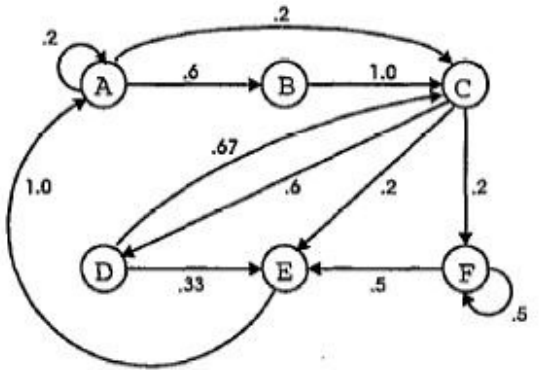


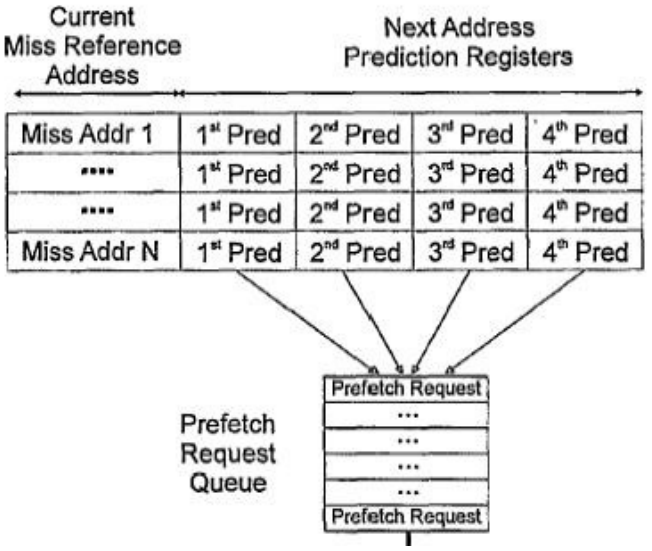
# Markov prefetcher

- Uses past history's reference patterns to determine which address to prefetch next
- Markov Predictor uses the history to build a state transition table of the next prefetched addresses and their statistical probabilities

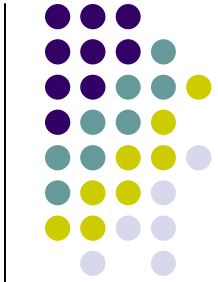
A, B, C, D, C, E, A, C, F, F, E, A, A, B, C, D, E, A, B, C, D, C

Figure 2: Sample miss address reference string. Each letter indicates a cache miss to a different memory location.



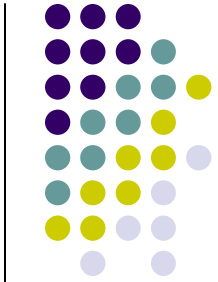



# Our motivation



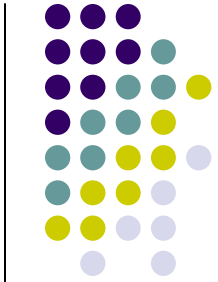
- The deficiency of the Markov prefetcher implemented in the paper is that with increased number of next states or increased table size, you end up fetching more and the memory bandwidth goes up dramatically, compared to the accuracy improvement.

# Our hypothesis



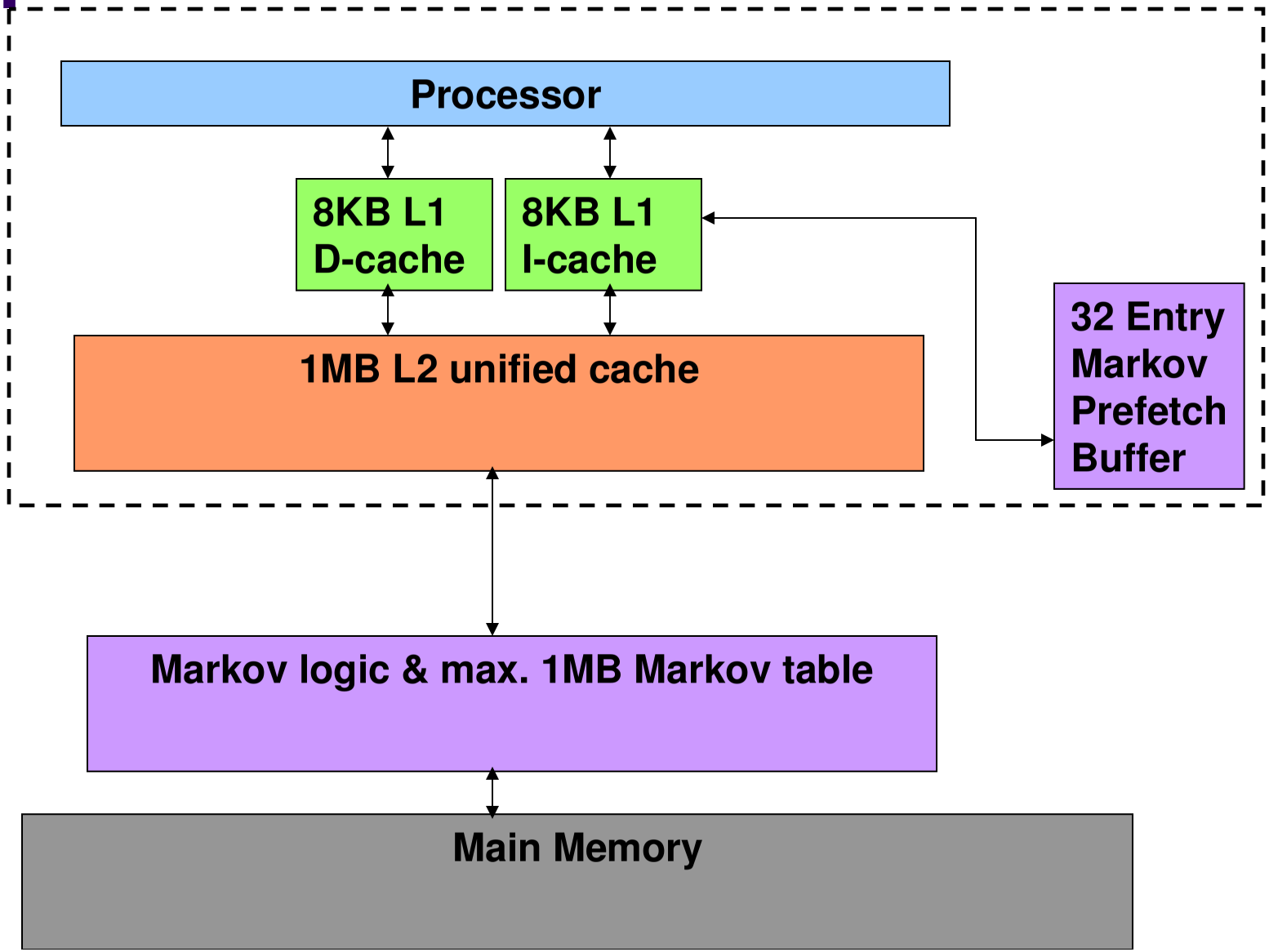
- We can modify the Markov model (by adding more states or a stride buffer before it) and get better prefetch accuracy, but also reduce required bandwidth at the same time

# Architecture



- Separate L1 instruction and data caches, 32kb each
- Unified 1Mb L2 cache
- Markov prefetch table & prefetcher logic inserted between L2 and main memory
- Markov prefetch buffer of 32 entries at same level as L2

# Architecture of Markov prefetcher



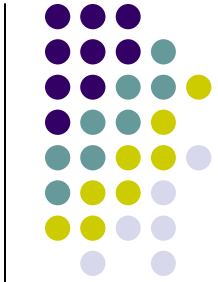
# First step – to prove Markov can be made better

- Wrote Markov prefetcher in C
- Compiled and ran this prefetcher standalone with SPEC2000 benchmark trace as input
- How did we generate traces from the benchmarks?
  - Modified sim-cache.c to printout L2 data and instruction cache miss addresses and dump it in a file
  - Modified default configuration file to match our memory hierarchy
  - Used the following benchmarks chosen: ammp, crafty, gcc, gzip, mcf, vpr

# Summary of results – Markov alone

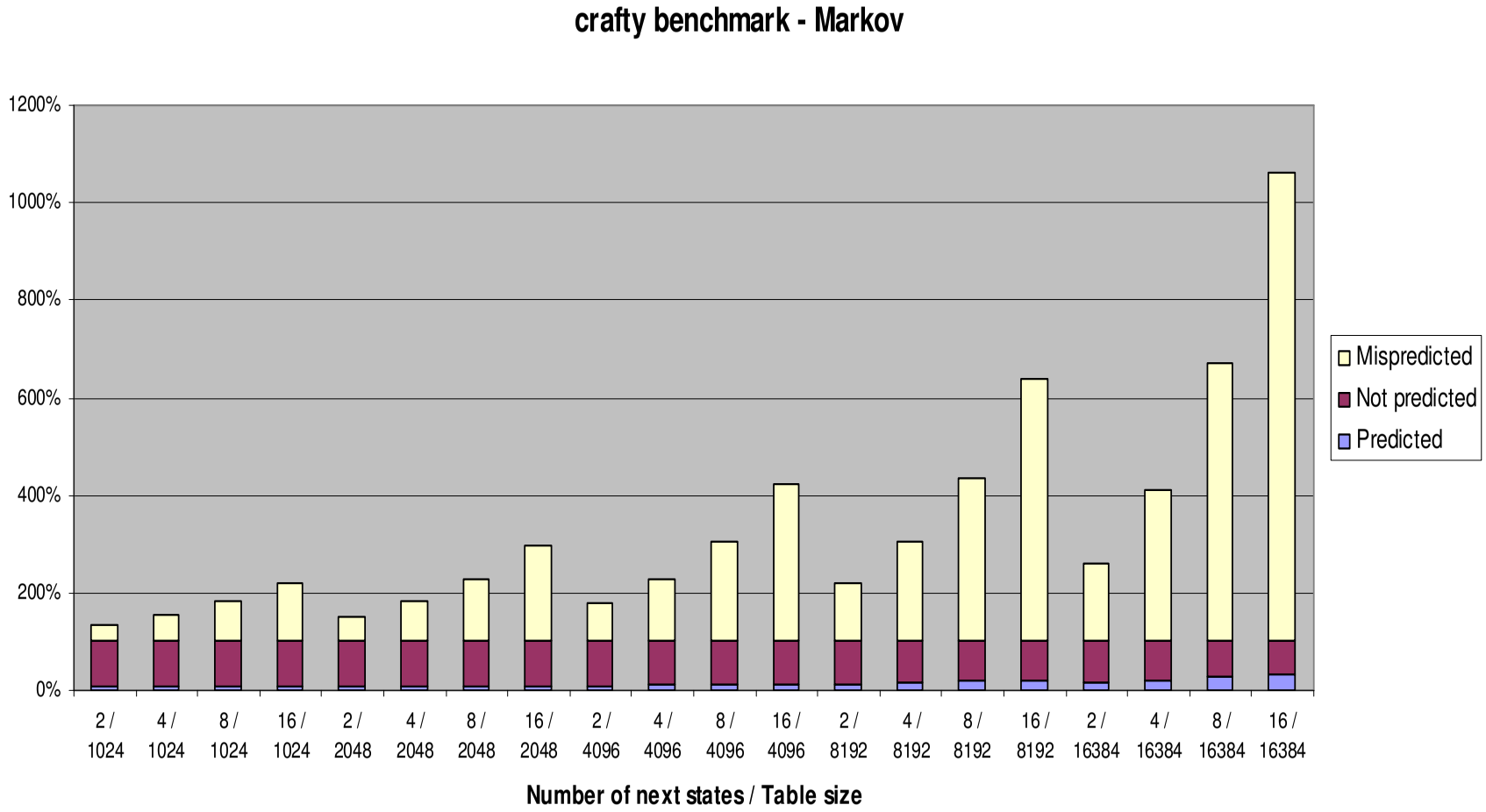
- The Markov prefetcher only fetches next states that are valid (non-zero).
- Increasing number of next states increases misprediction
- Increasing table size increases prediction (coverage)
- With Markov alone, increasing the table size and number of next states increases the accuracy, but not enough to justify the high misprediction rate
- When we add a stride prefetch buffer before the Markov, the accuracy is still high, but the misprediction drops dramatically
- If we limit the number of next states prefetched to half for the combined Markov and Stride prefetchers, the misprediction doesn't go down

# Definitions – Markov alone

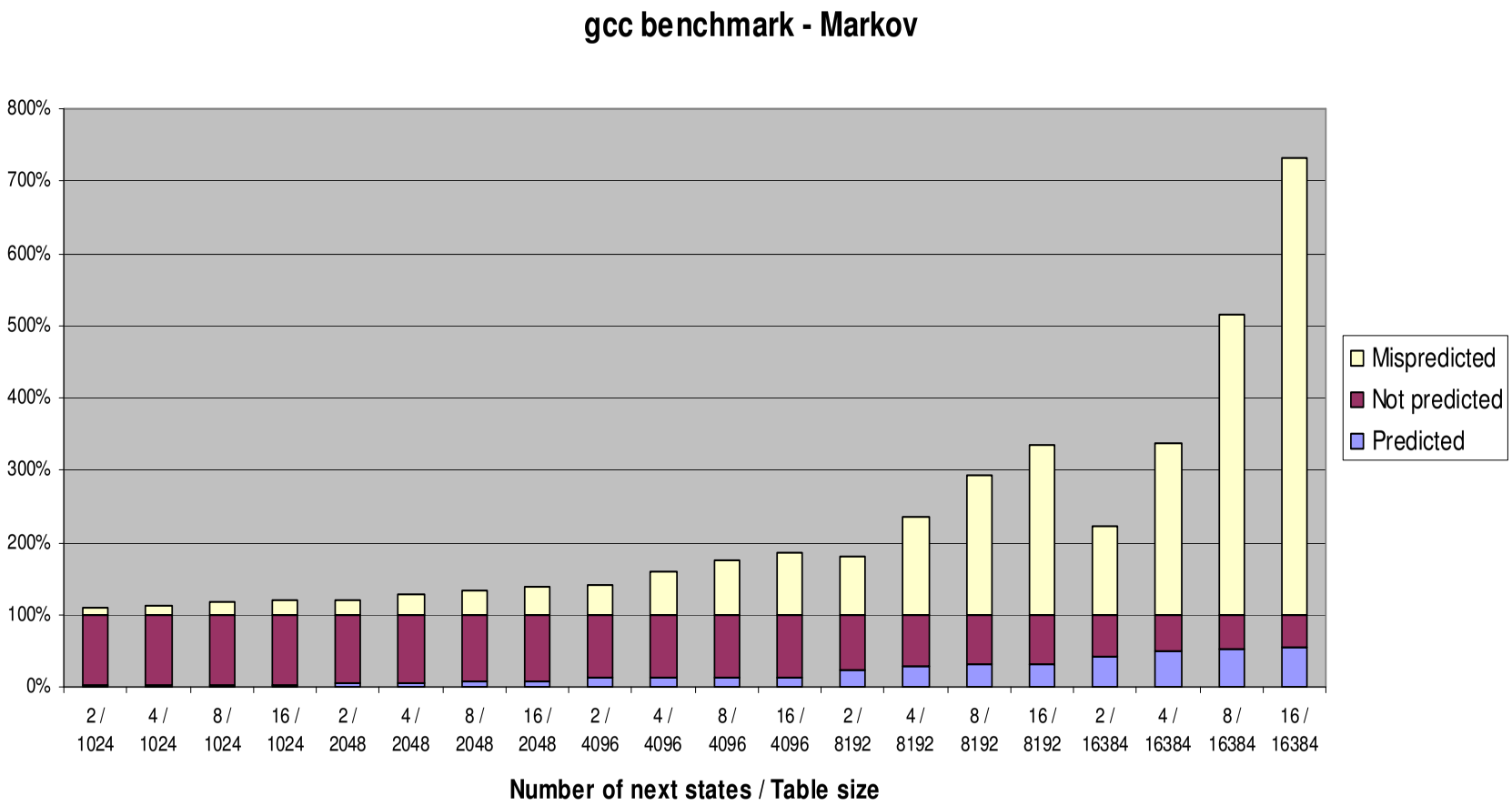


- Predicted = Coverage = Markov prefetch buffer hits / Total demand fetches
- Not predicted = 1 – Predicted
- Mispredicted = Total wasted prefetches / Total demand fetches

# Influence of number of next states – Markov alone



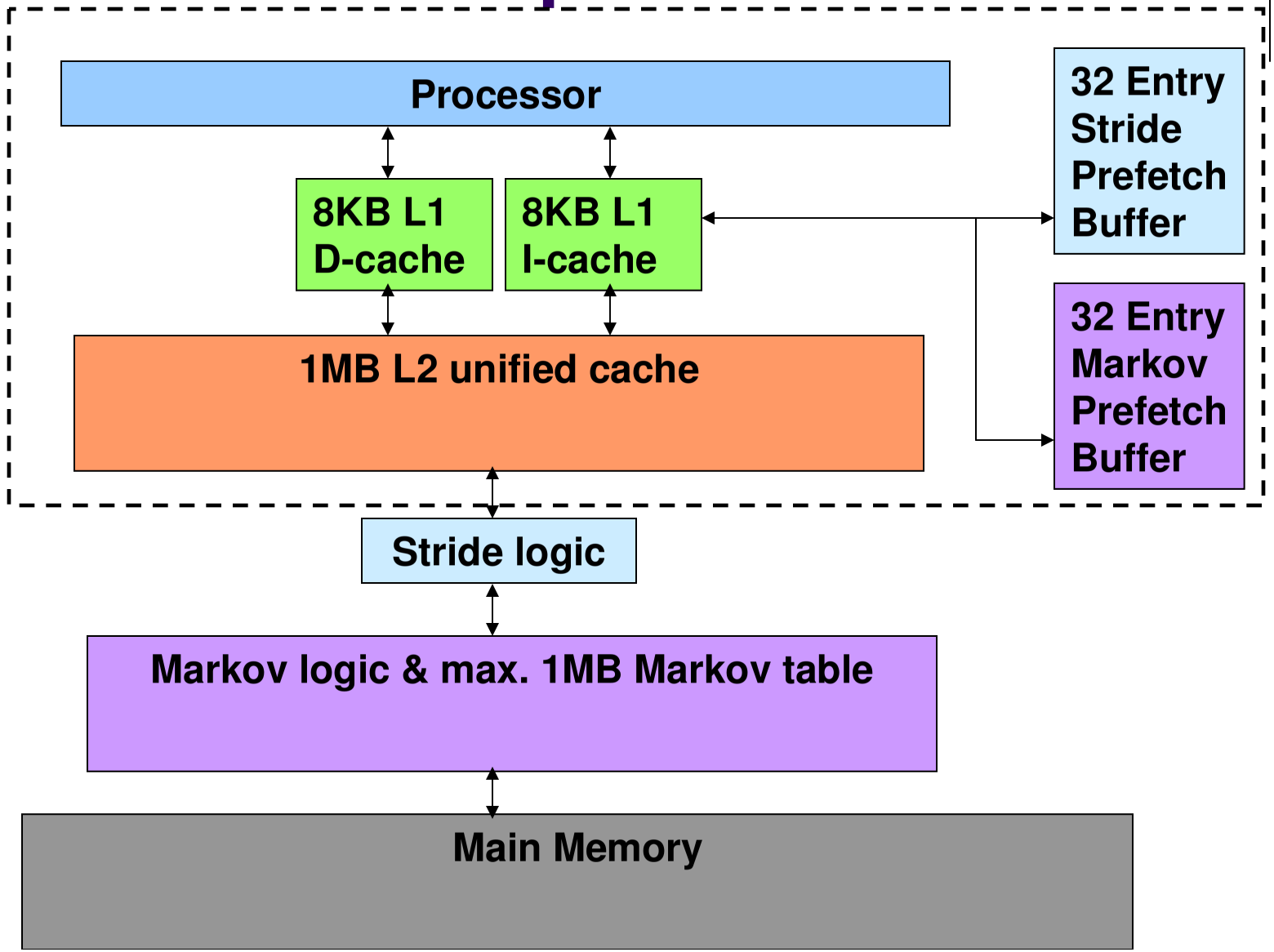
# Influence of number of table size – Markov alone



# Why combine Markov with Stride?

- Markov doesn't predict one-time sequential accesses
- Markov alone has high amount of wasted prefetches
- Markov catches larger prefetch distances once you filter out strided accesses
- Implement single stride prefetcher instead of multiple stride and stream prefetchers

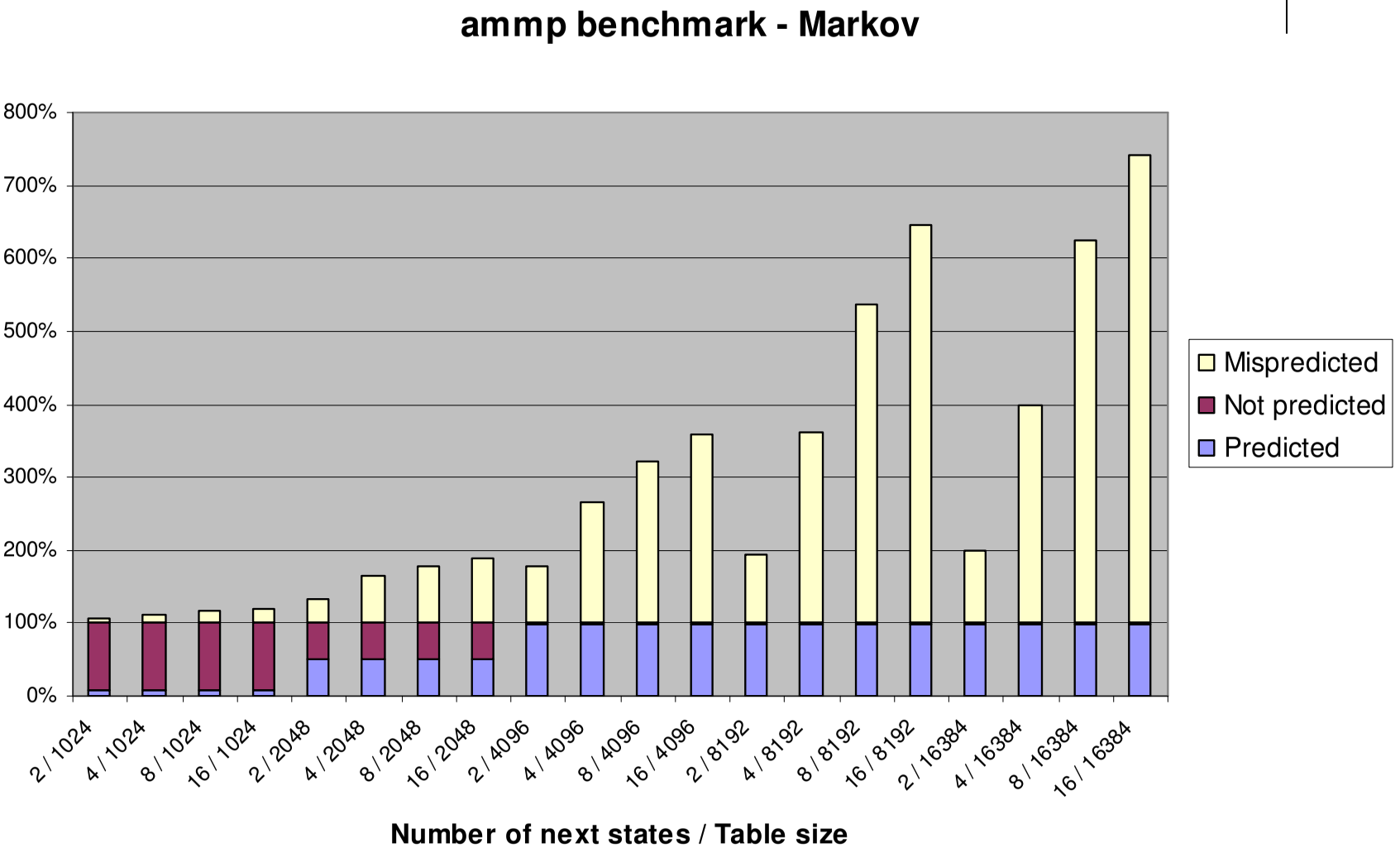
# Architecture of combined Markov/Stride prefetcher



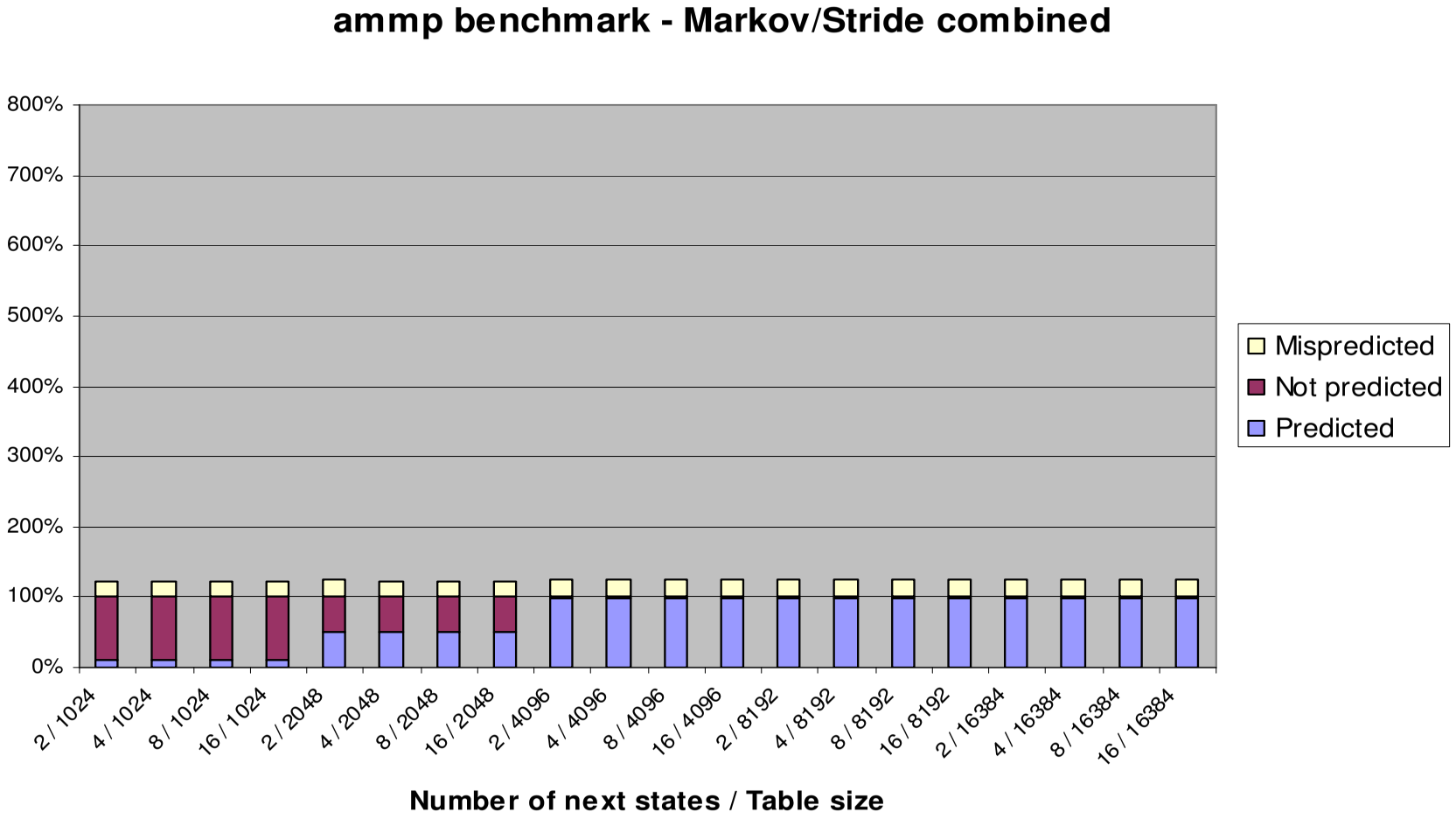
# Definitions – Markov and Stride combined

- Predicted = Coverage = (Markov prefetch buffer hits + Stride prefetch buffer hits) / Total demand fetches
- Not predicted = 1 – Predicted
- Mispredicted = (Total wasted Markov prefetches + Total wasted Stride prefetches) / Total demand fetches

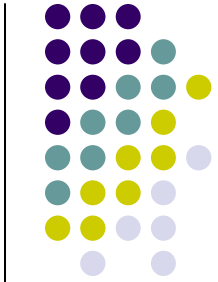
# Markov alone – Markov catches most - #1

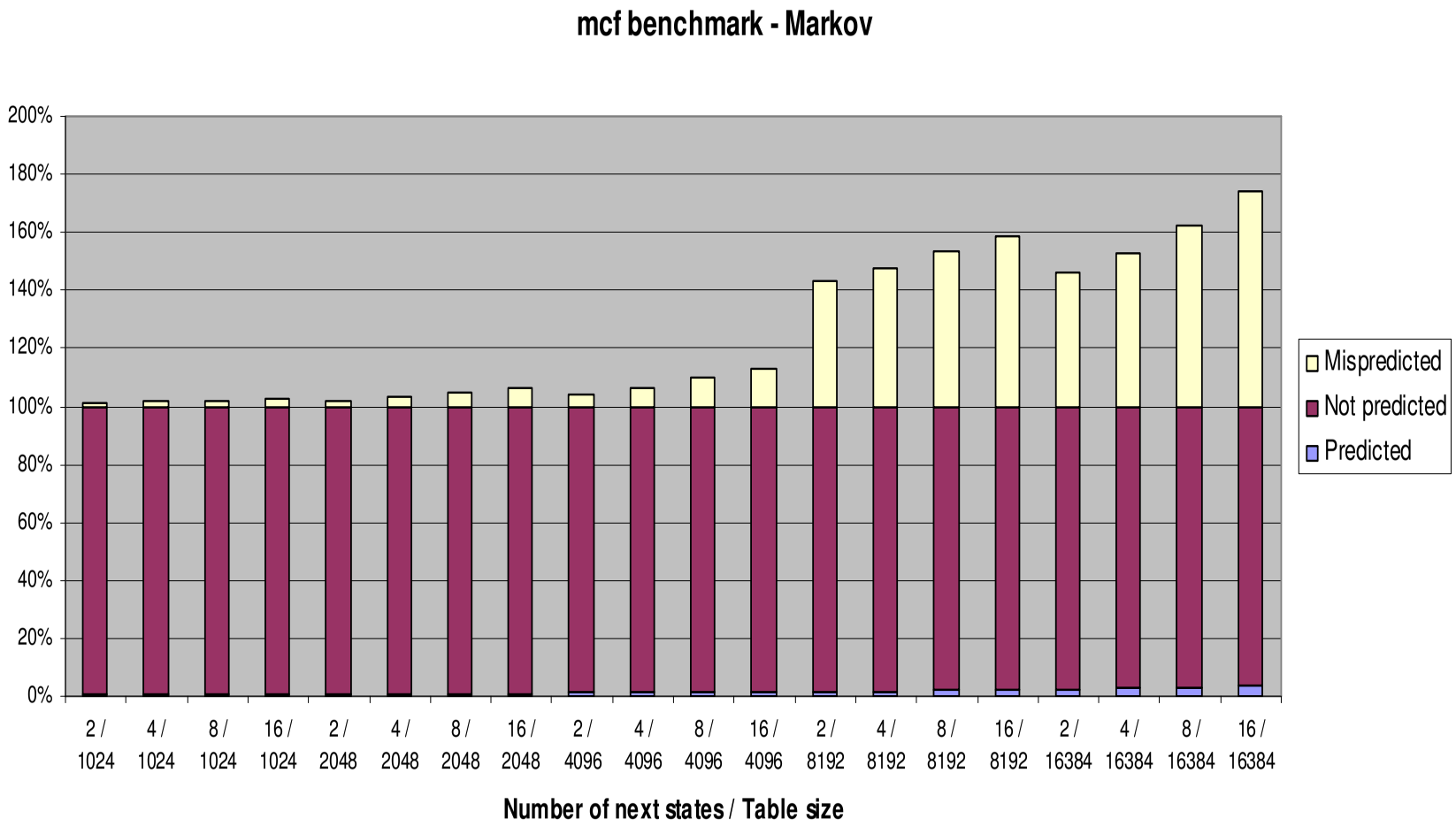


# Markov/Stride combined – Markov catches most - #2

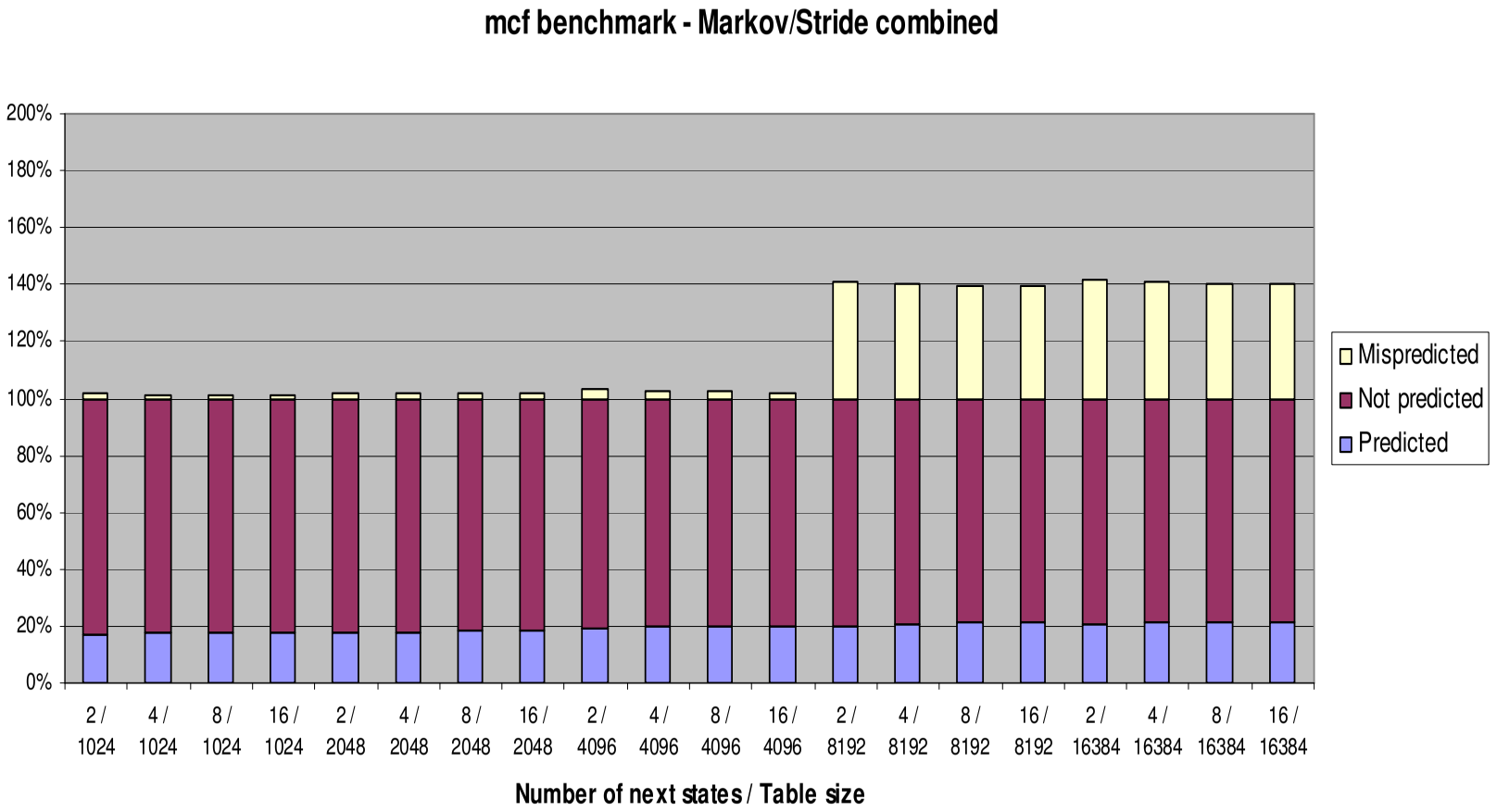


# Markov alone – Stride catches most - #1

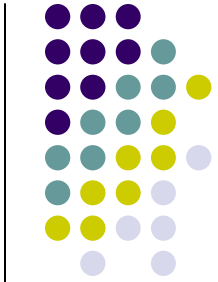




# Markov and Stride combined – Stride catches more - #2

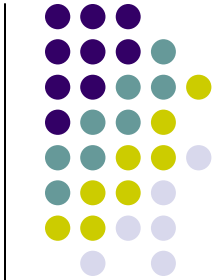


# Area cost



- We estimate that the majority of the area is taken up by the Markov table
- Our largest Markov table is 16 next states with 16k entries, which is 1Mb.  Compared to Markov paper that recommended two 64K entry by 4 next state tables (2MB of table space)
- This is similar to adding another 1Mb of L2 on-chip

# Conclusion



- Markov prefetching is still useful beyond the L1 cache for certain applications.
- Simple stride prefetcher in series with Markov prefetcher can dramatically reduce mispredicted prefetches
- Implementation requirements for prefetcher can be smaller than previously proposed with similar performance
- Full performance impact not yet calculated
- Might not be suitable for SMT applications. May be implemented with separate tables with for different threads