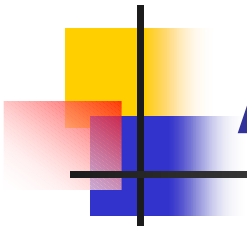
MATCH: Memory Address Trace CacHe

Arthur Nieuwoudt
David Leal
Noah Deneau
Michael Haag

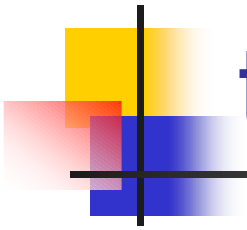
04/22/2003



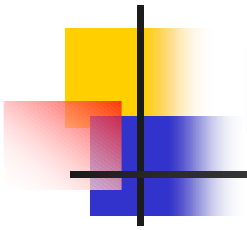
Agenda

- Motivation
- Hypothesis
- Architecture
- Methodology
- Analysis
- Conclusions
- Future Work

Motivation – Breaking Down the Memory Wall

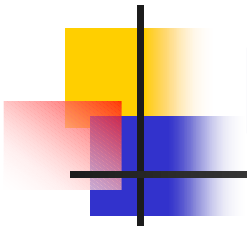


- For wide-issue superscalar processors, memory latency is increasing relative to cycle time
- Not all memory accesses exhibit the spatial locality that standard caches exploit
- Other memory access patterns could exist



Related Concept: Trace Cache

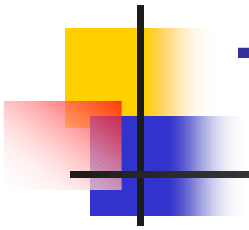
- Instruction trace caches are used to exploit patterns in instructions
  - Research has demonstrated substantial performance improvements
  - Used in current processor designs
- Can this concept be extended to data memory?



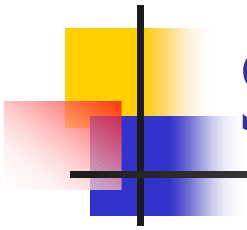
Hypothesis

- Memory Access patterns exist that cannot be exploited by standard caches
- We can capture these patterns in reasonably sized data structures
- By capturing these patterns, we will improve performance on a wide range of applications

Memory Trace Analysis: Are There Patterns Out There?



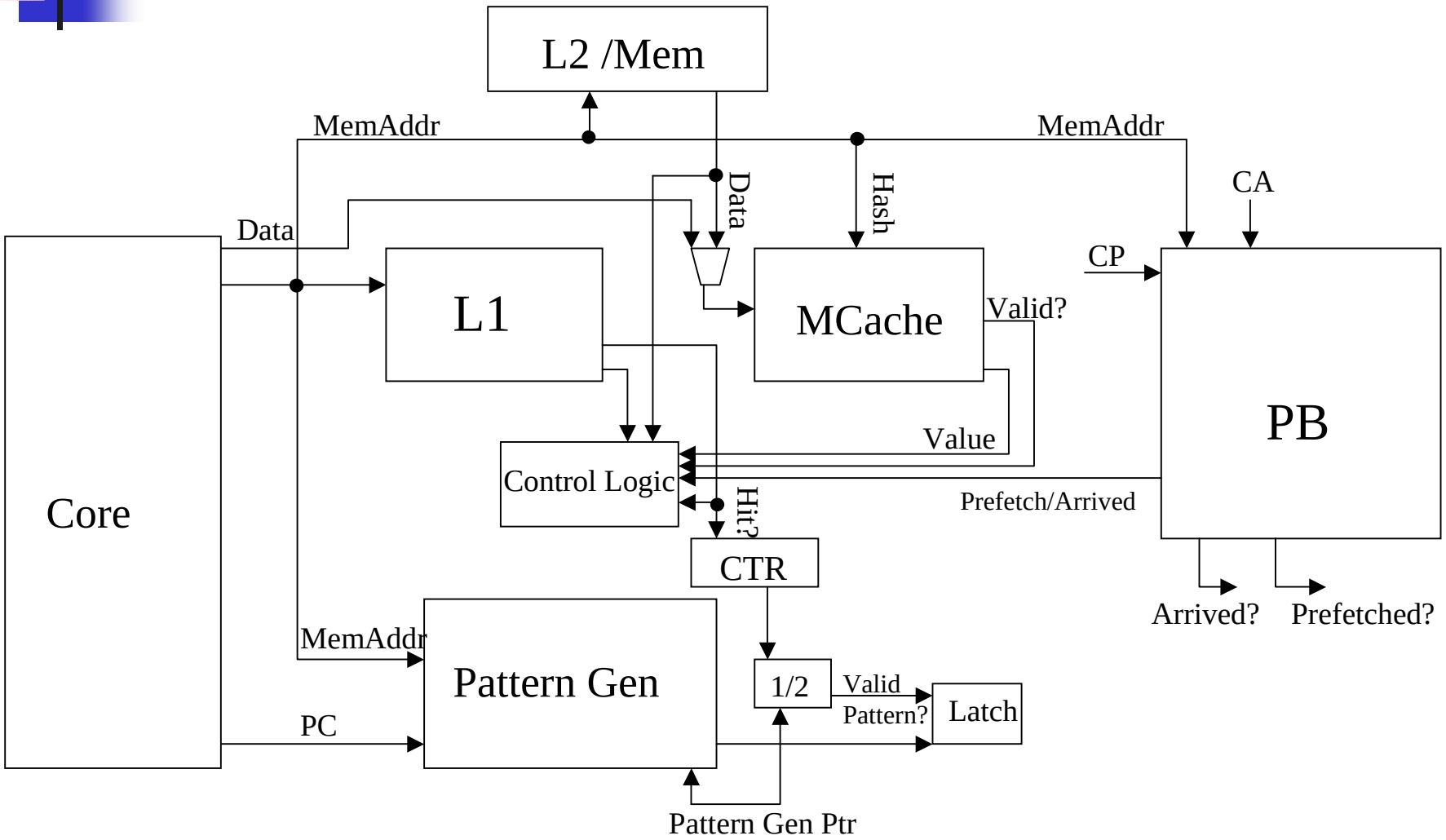
- Analyzed 5 SPEC2000 benchmarks
- Used several different pattern generation algorithms
- Evaluated statistics such as frequency, length, utilization
  - Patterns exist with sufficient length and frequency to exploit

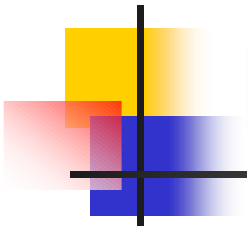


Solution: MATCH Architecture

- Pattern Generator
  - Generates patterns of memory accesses that poorly utilize the L1 data cache
- Pattern Buffer
  - Stores traces of memory addresses generated by the Pattern Generator
  - Prefetch data from L2 cache and main memory
- MATCH Cache (M-Cache)
  - Stores data associated with access patterns

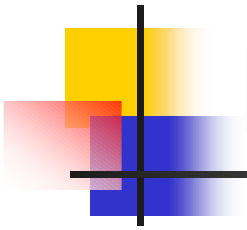
MATCH: The Gory Details





Methodology

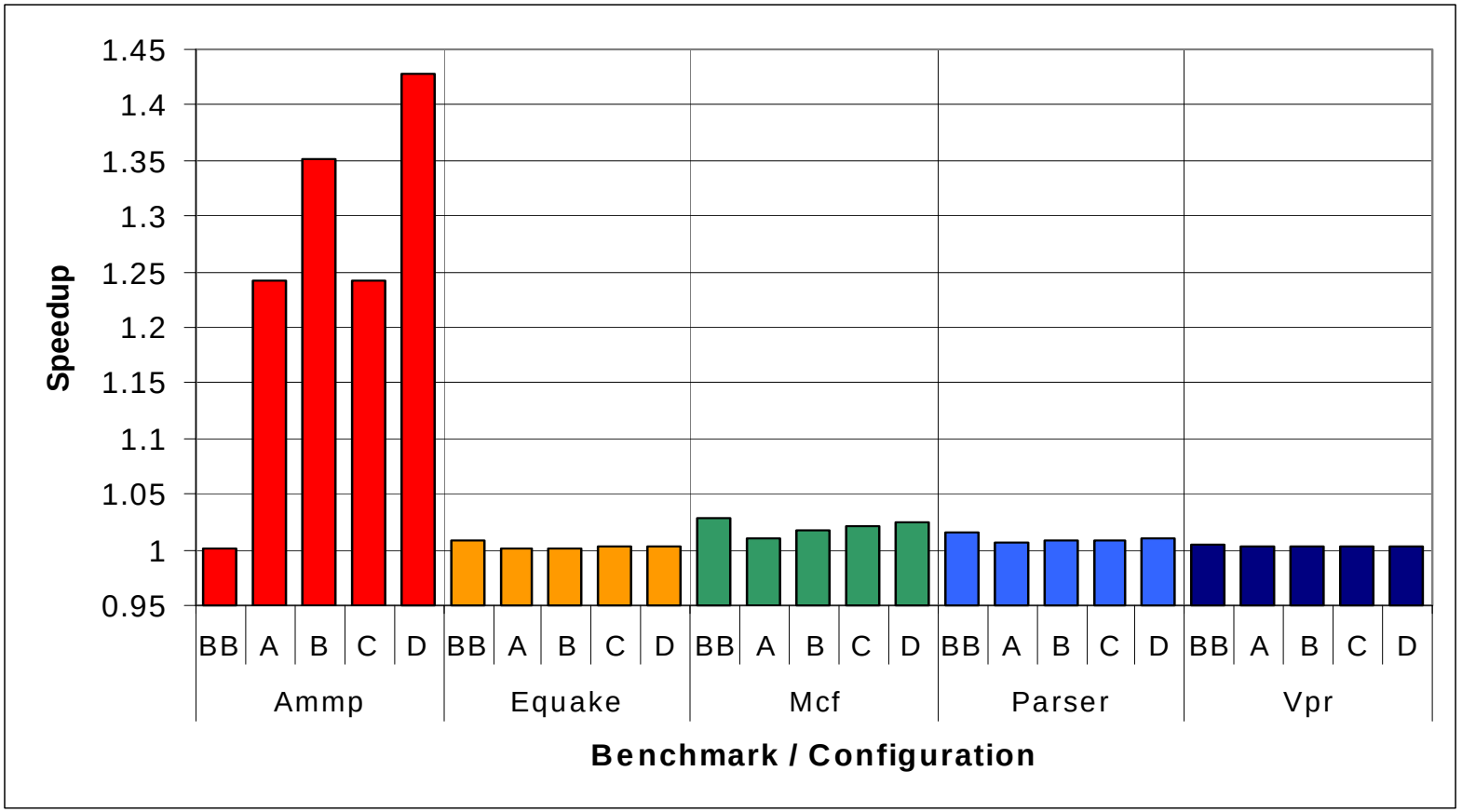
- Modified Simplescalar cache functions to simulate the MATCH architecture
- Baseline configurations:
  - Baseline – 8 KB L1 Data Cache, no MATCH
  - Big Baseline (BB) – 16 KB L1 Data Cache, no MATCH



MATCH Configurations

- All tested configuration have 8 KB L1 data cache
- Tested MATCH configurations
  - A – 8 KB M-Cache, 8 KB Pattern Buffer
  - B – 8 KB M-Cache, 64 KB Pattern Buffer
  - C – 16 KB M-Cache, 8 KB Pattern Buffer
  - D – 16 KB M-Cache, 64 KB Pattern Buffer

Results

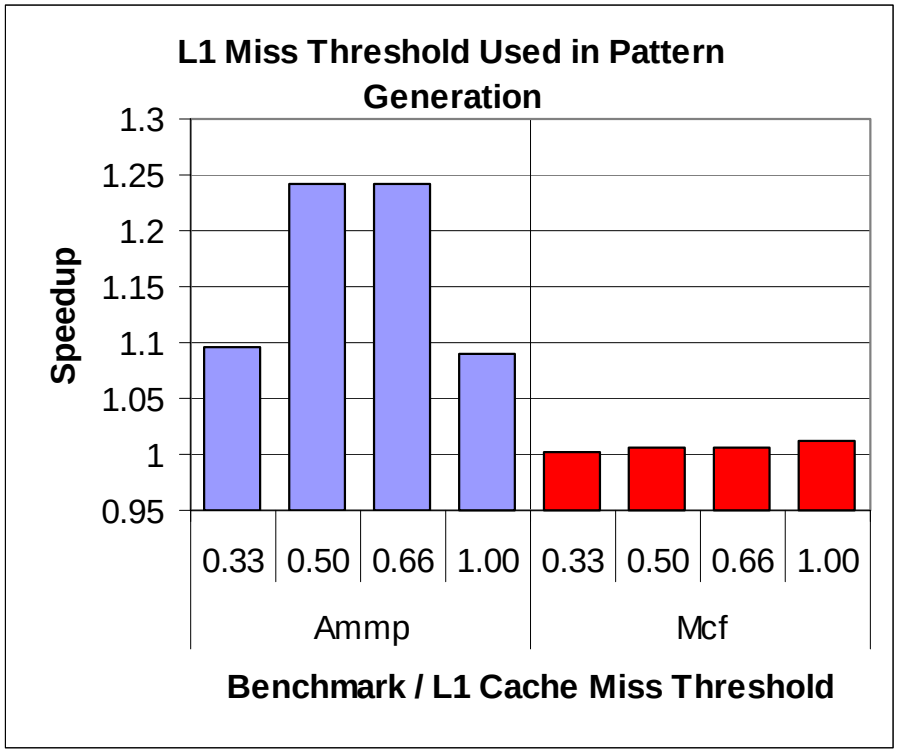

1.45
1.4
1.35
1.3
1.25
1.2
1.15
1.1
1.05
1
0.95
Speedup
BB A B C D BB A B C D BB A B C D BB A B C D BB A B C D
Ammp
Equake
Mcf
Parser
Vpr
Benchmark / Configuration

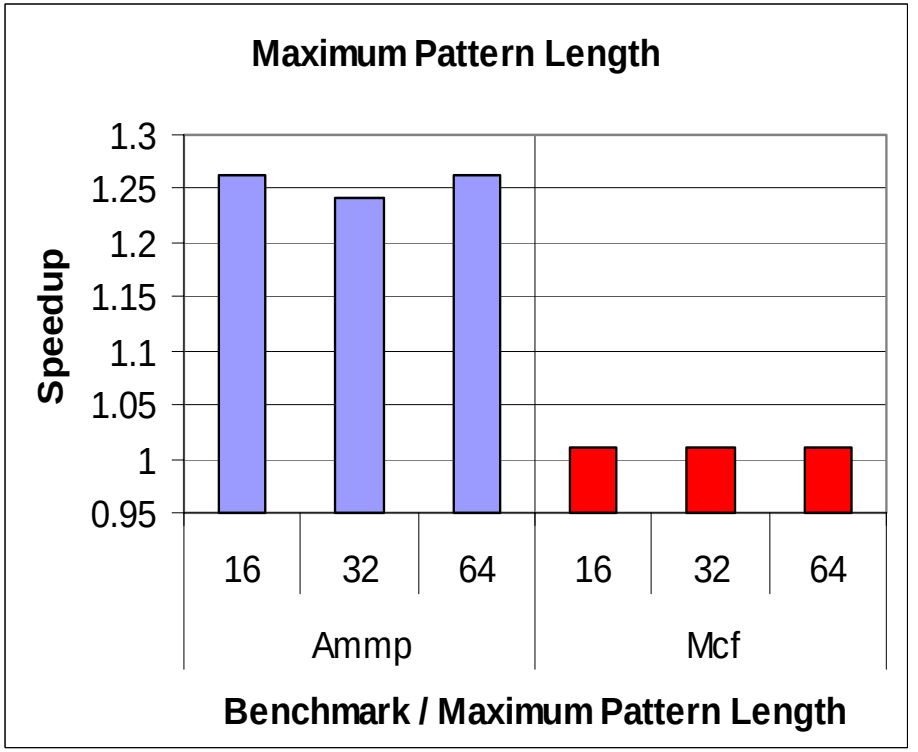
L1 Miss Rate and Performance

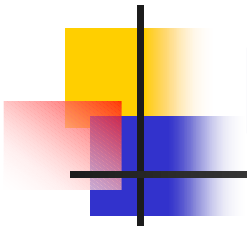
- MATCH performance depends on L1 miss rate
- Normalized performance provides a estimate of efficiency

Application	L1 Miss Rate	MATCH Speedup / Miss Rate
Ammp	0.236	1.815
Equake	0.033	0.077
Mcf	0.136	0.185
Parser	0.034	0.302
Vpr	0.012	0.232

MATCH Configuration Results

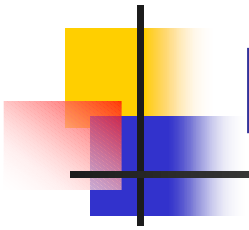






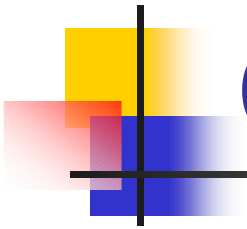
Revised Hypothesis

- MATCH works well in specific programs
  - High L1 miss rate
  - Poor spatial locality
  - Repeated work on fixed data sets
- Patterns do exist
- Reasonably sized data structure are effective for *certain* applications
- Requires further research



Future Work

- Does Ammp's performance exist in other applications?
  - Scientific applications may be a good starting point
- Evaluate other pattern generation schemes
- More efficient Pattern Buffer



Questions / Comments