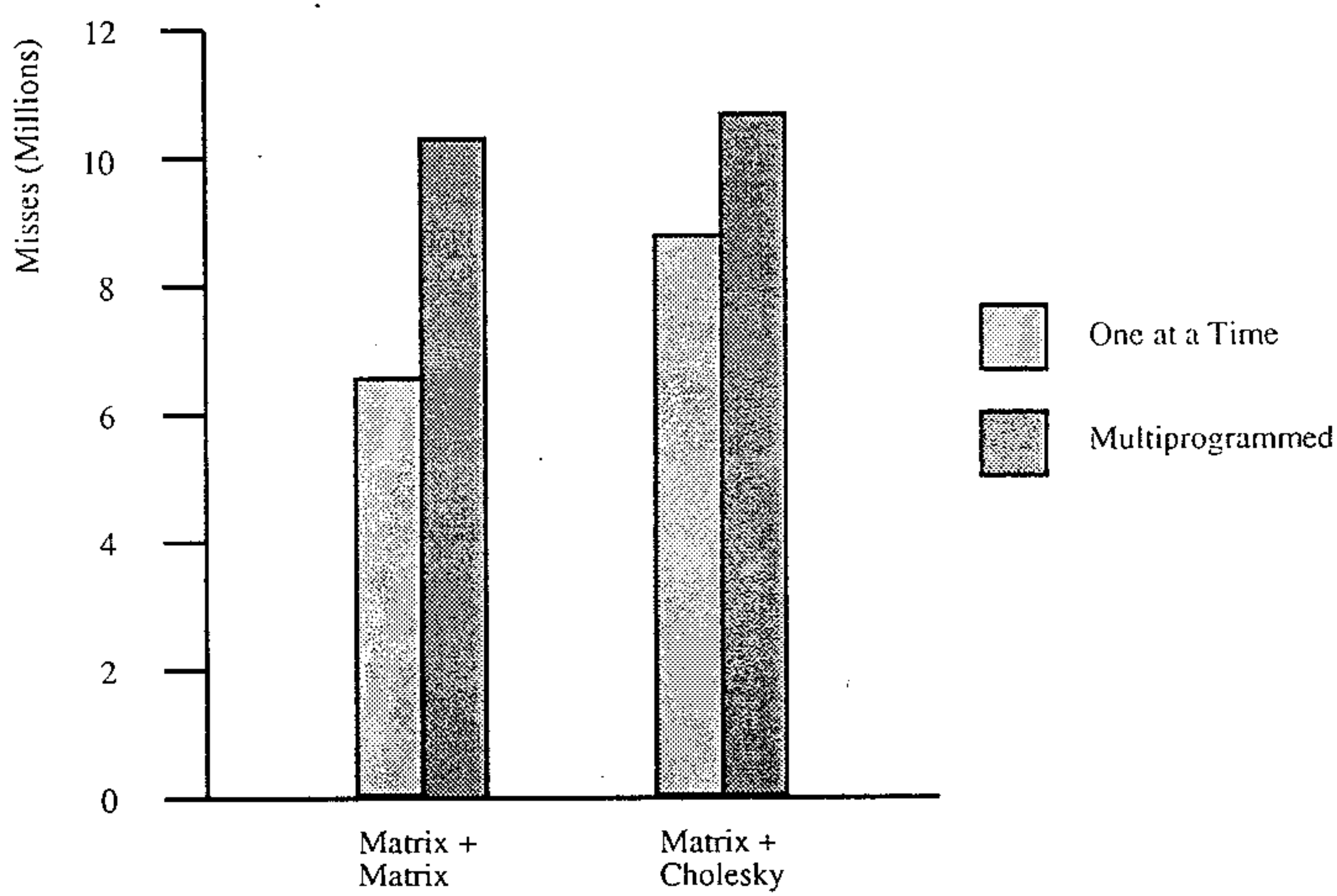
Evaluating the Benefits of Cache-Affinity Scheduling in Shared-Memory Multiprocessors

Multiprogramming Misses



Outline

- Experimental environment & workloads
- Potential benefits of cache affinity
- Implementation of affinity scheduling
- Performance results

Experimental Environment

- 4-processor Silicon Graphics workstation (33MHz R3000 CPU)
  - Shared memory
  - 64 Kbyte I-cache
  - 64 Kbyte + 256 Kbyte D-cache
  - 30-cycle cache miss penalty
- IRIX operating system (UNIX System V)
- Hardware monitor tracks cache misses

Parallel Applications

- Matrix: Blocked matrix multiply
- Cholesky: Cholesky factorization of sparse matrices
- Mp3d: Particle simulator
- Pmake: Parallel compilation of 24 C files with 460 lines
- Oracle: Cached TP1 benchmark on an Oracle database

... grouped in workloads

Potential Benefits of Cache Affinity

- Application characteristics
  - Amount of reused cache state
  - Time between potential re-schedules *(Dispatch Interval)*
  - Reason for terminating dispatch interval

- Interactions among the applications of the workload
  - Time between real re-schedules *(Effective Time Slice)*
  - State displaced by intervening applications

Potential Benefits of Cache Affinity (II)

Application	Distribution of the Dispatch Interval Duration (% of Total Intervals)						
	1–5ms	5–9ms	9–13ms	13–17ms	17–21ms	21–25ms	25–30ms
Matrix	3	1	0	1	1	1	93
Cholesky	19	9	5	5	5	8	49
Mp3d	17	9	8	5	14	11	36
Pmake	40	11	8	6	7	8	20
Oracle	67	32	1	0	0	0	0

Cause	*Matrix* (%)	*Cholesky* (%)	*Mp3d* (%)	*Pmake* (%)	*Oracle* (%)
End of Quantum	94.6	46.2	26.2	21.5	1.1
Semaphore Block (I/O)	2.2	2.3	10.1	47.5	39.2
Synchronization	0.0	47.9	60.7	0.0	21.9
System Call	2.2	2.1	2.0	14.8	37.6
TLB Fault	0.0	0.2	0.2	10.3	0.0
Other	1.0	1.3	0.8	5.9	0.2

Promising Workloads

- Reuse cache state
- Short executions before preemption
- Block infrequently
- Interleaved with processes that displace state

Workload	Misses in Cache Reload (Thousands)	Median Eff. Time Slice (ms)	Frequent Process Block?	Misses of Intervening Application (Thousands)	Potential Affinity Benefit
Matrix+Matrix	3+3	30+30	N+N	6+6	mod.+mod.
Matrix+Cholesky	3+1	30+25	N+N	6+5	mod.+low
Matrix+Mp3d	3+0	30+20	N+N	6+18	mod.+none
Pmake+Matrix	0.5+3	5+30	Y+N	1.5+6	low+low
Pmake+Cholesky	0.5+1	5+25	Y+N	1.5+5	low+low
Pmake+Pmake	0.5+0.5	5+5	Y+Y	1.5+1.5	low+low
Oracle	2	3	Y	2.5	mod.

Cache Affinity Scheduling

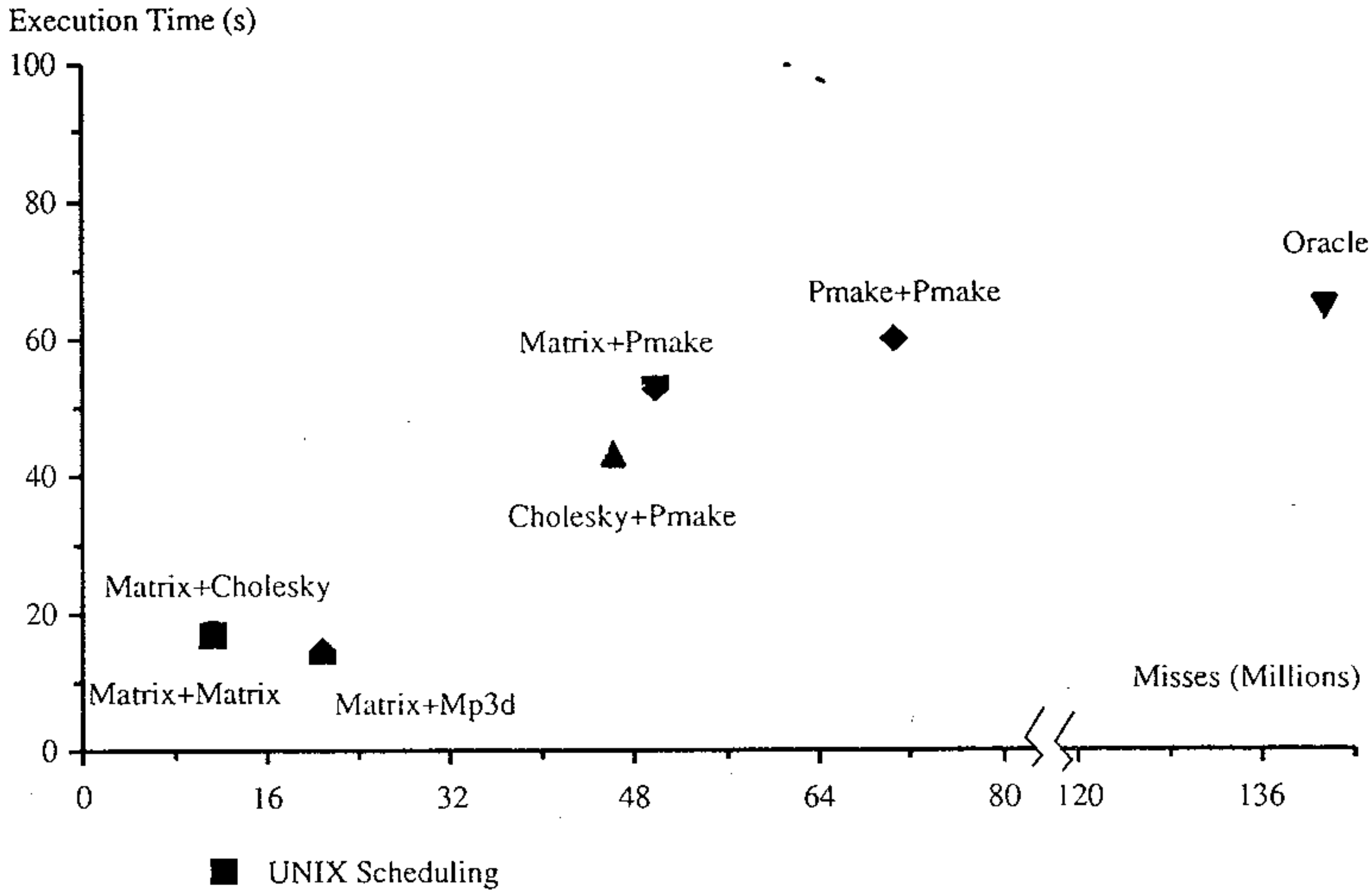
- Minimize
  - Cache state displacement
  - Process migration

- However, still keep
  - Load balance
  - Fairness
  - Response time

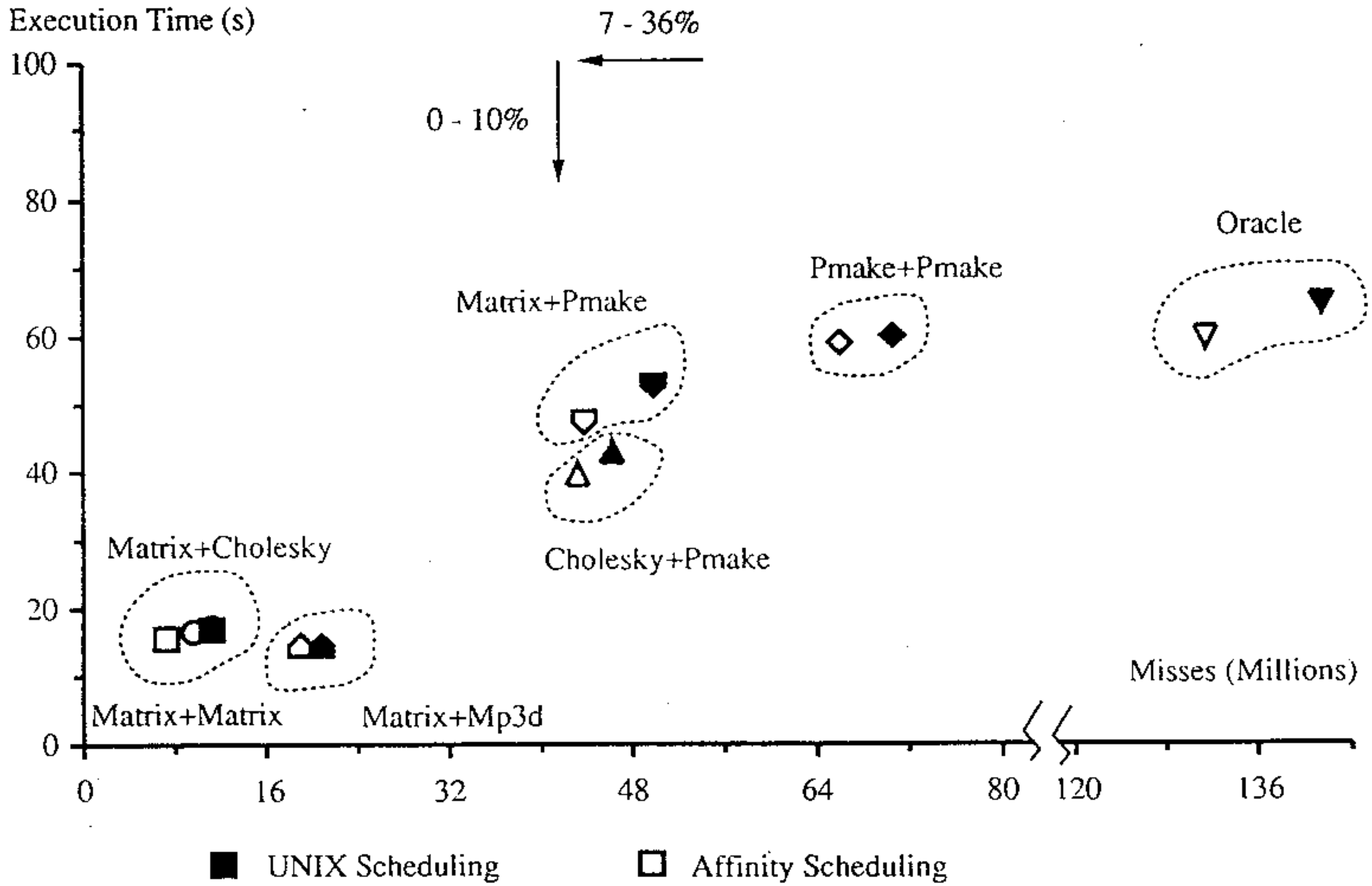
Implementation

- Based on standard UNIX priority scheduling
  - Priority decreases as process accumulates CPU time
- Some priorities *temporarily* increased with affinity
  - Process that just ran on scheduling CPU
  - Processes whose most recent execution was on CPU

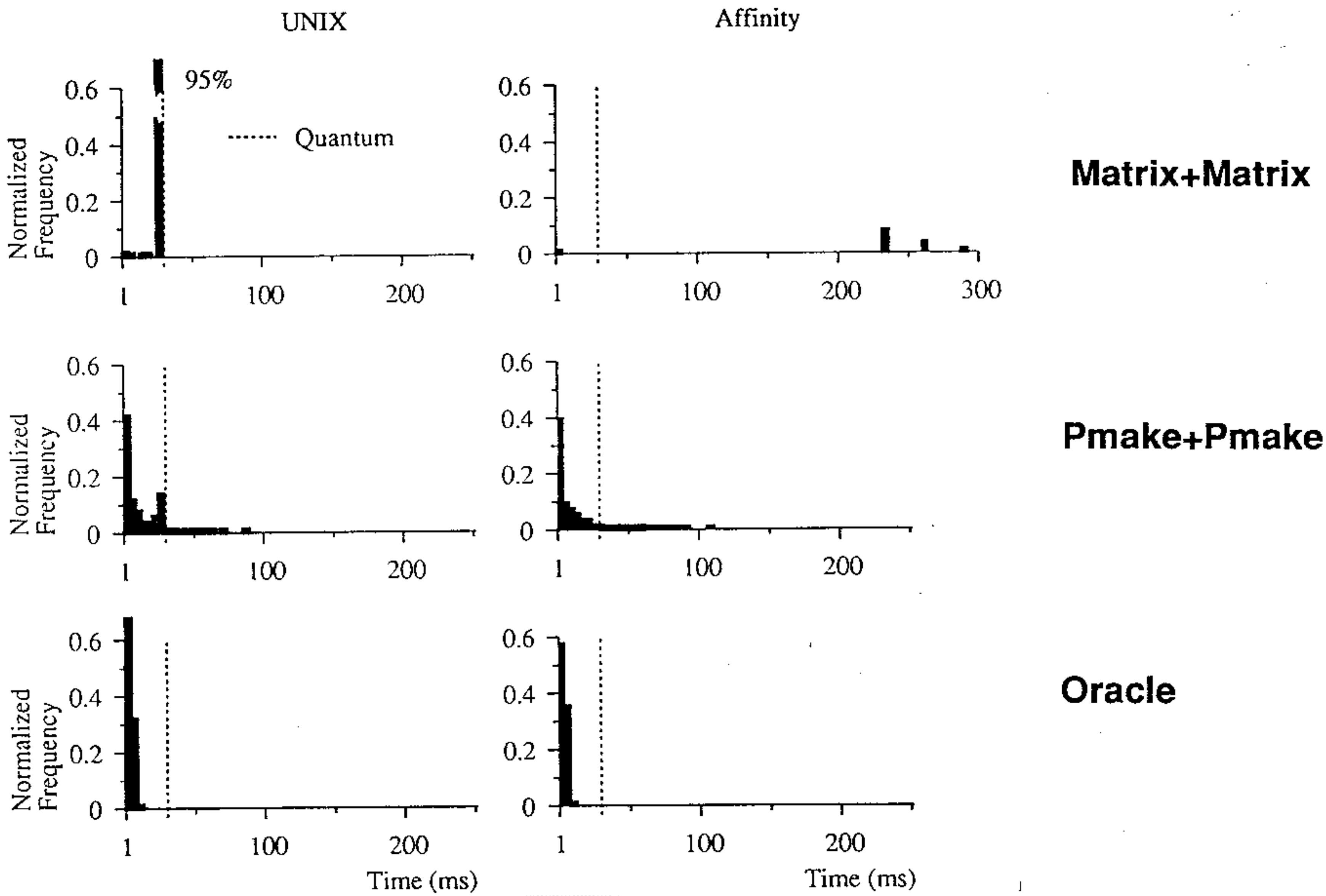
Workloads Studied

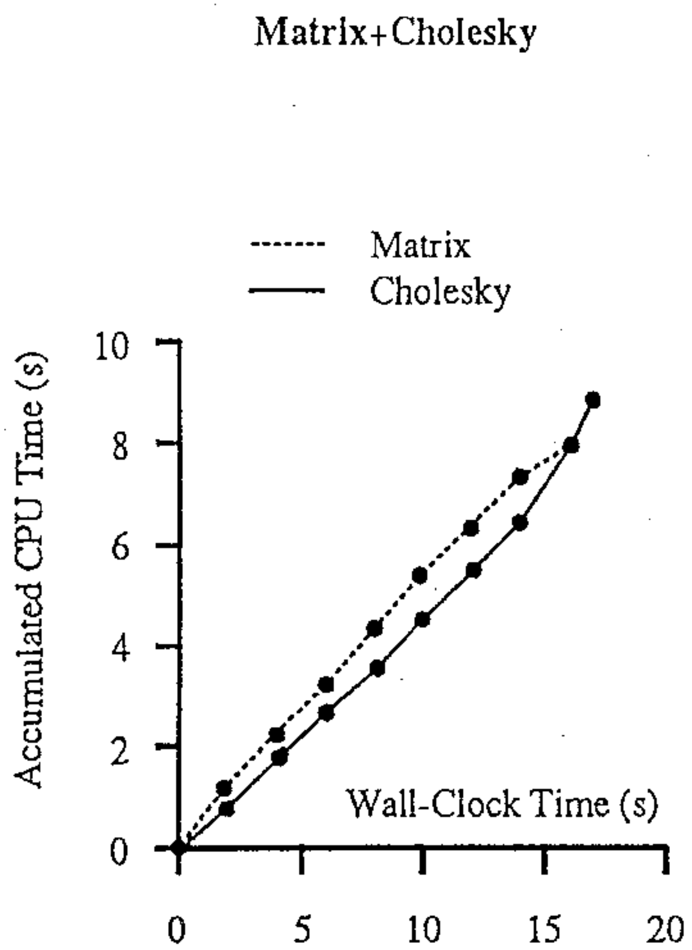


Affinity Scheduling Performance

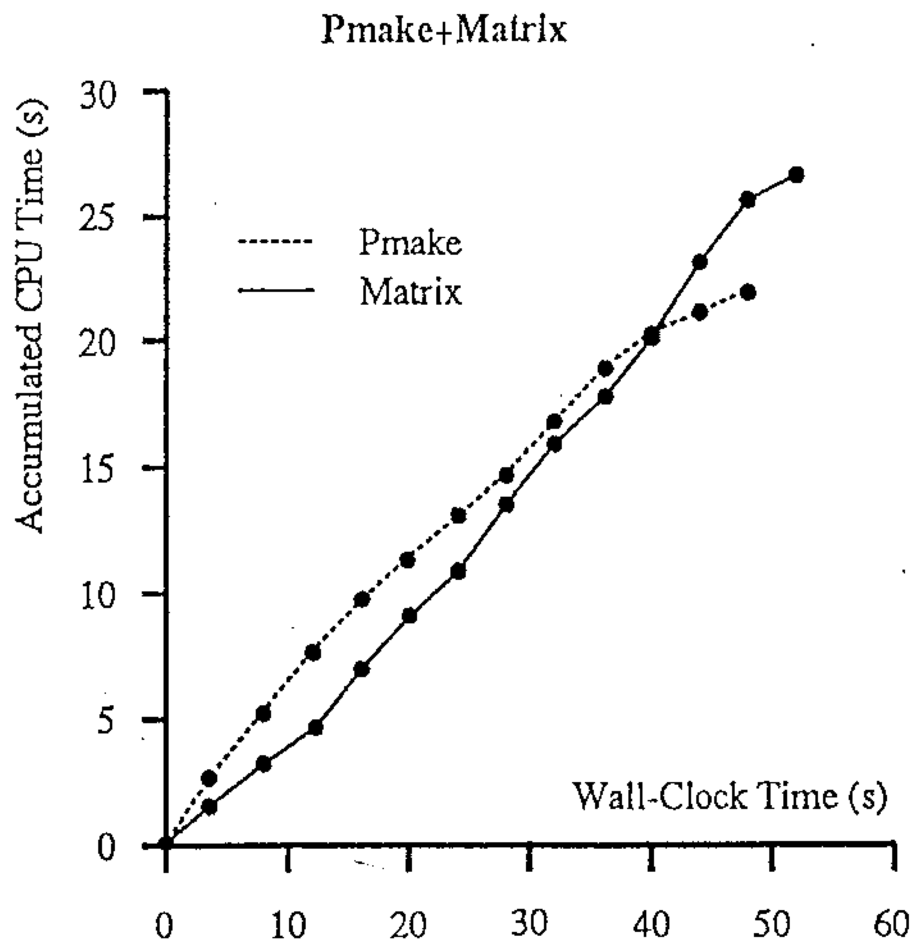


Time Between Re-Schedules





Matrix+Cholesky
Matrix
Cholesky
Accumulated CPU Time (s)
Wall-Clock Time (s)
0
2
4
6
8
10
0
5
10
15
20



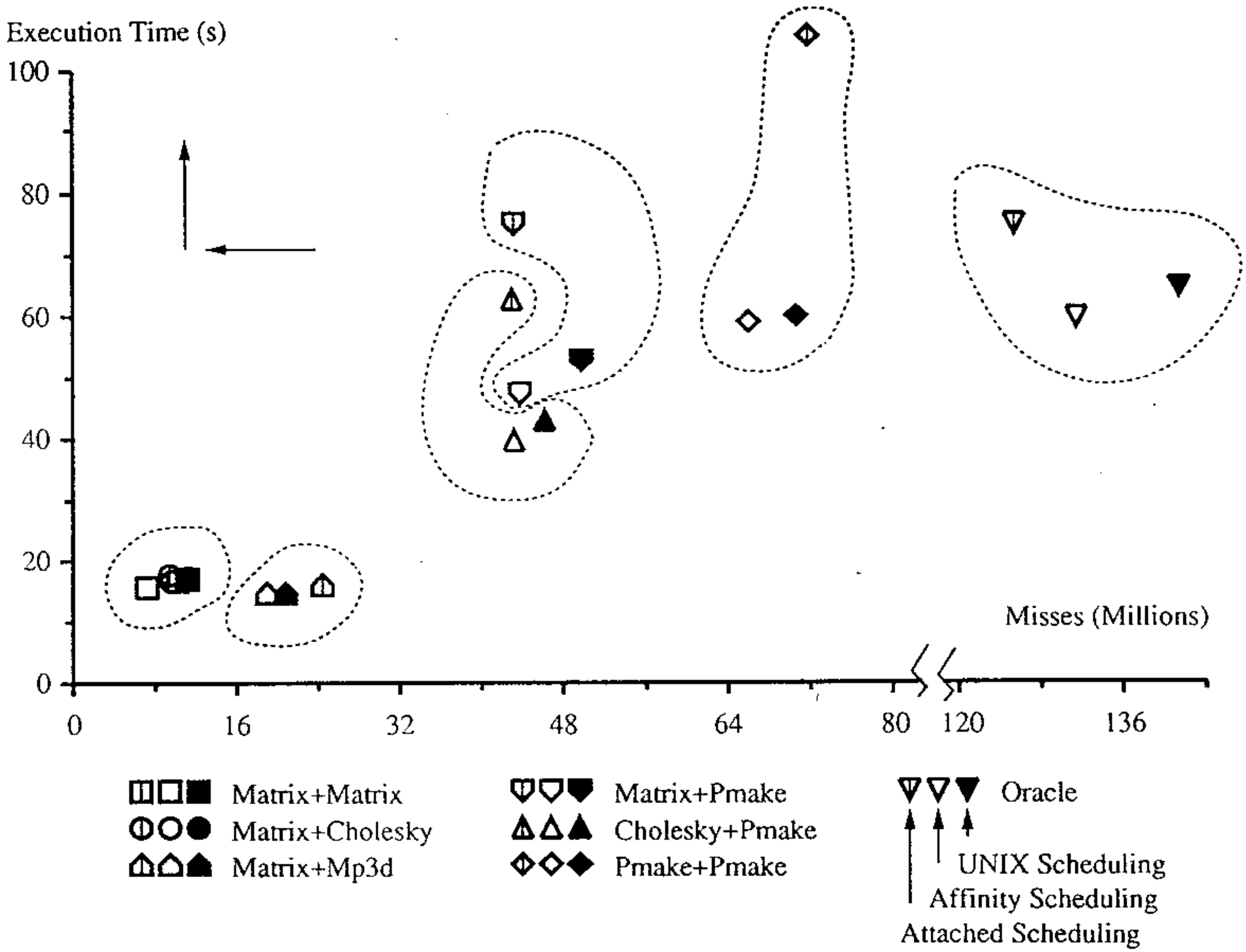
Pmake+Matrix
Pmake
Matrix
Accumulated CPU Time (s)
Wall-Clock Time (s)
0
5
10
15
20
25
30
0
10
20
30
40
50
60

• It is fair

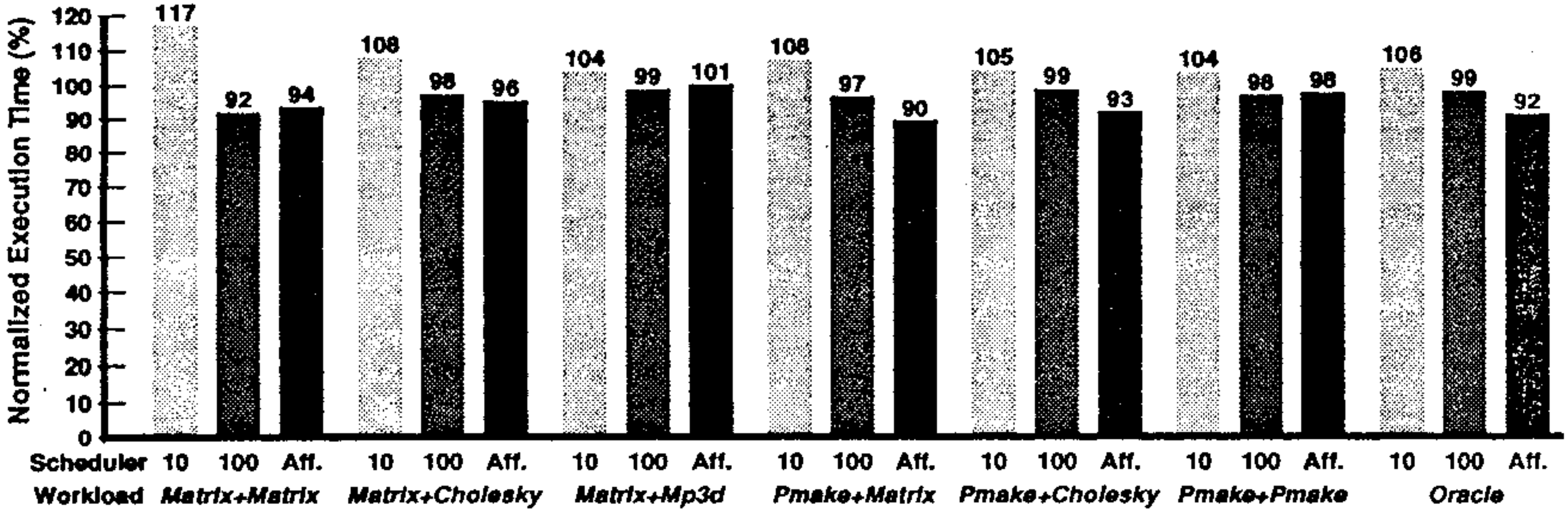
Alternative Approaches

- Attached or fixed scheduling
  - \+ No process migration
  - \- Cache state displacement
  - \- Load imbalance
- Long time quanta
  - \+ Reduced cache state displacement
  - \- Process migration (bad for processes that block)
- Process control
  - \- Unorthodox scheduling policy

Attached Scheduling



Effect of Time Quantum Duration



- **Affinity scheduling is better than long quanta**

Summary

- Cache affinity scheduling reduces misses significantly
- Performance impact is small but positive
- Simple implementation is fair and effective
- Alternatives not as robust