

# Architecture Optimizations for Synchronization and Communication on Chip Multiprocessors

Sevin Fide and Stephen Jenks

Workshop on Multithreaded Architectures and Applications  
International Parallel and Distributed Processing Symposium  
April 18, 2008

UCIrvine

**SPeDS**  
Scalable Parallel and  
Distributed Systems Lab

# Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepushing
- Simulation Environment
- Simulation Results
- Conclusion

# Introduction

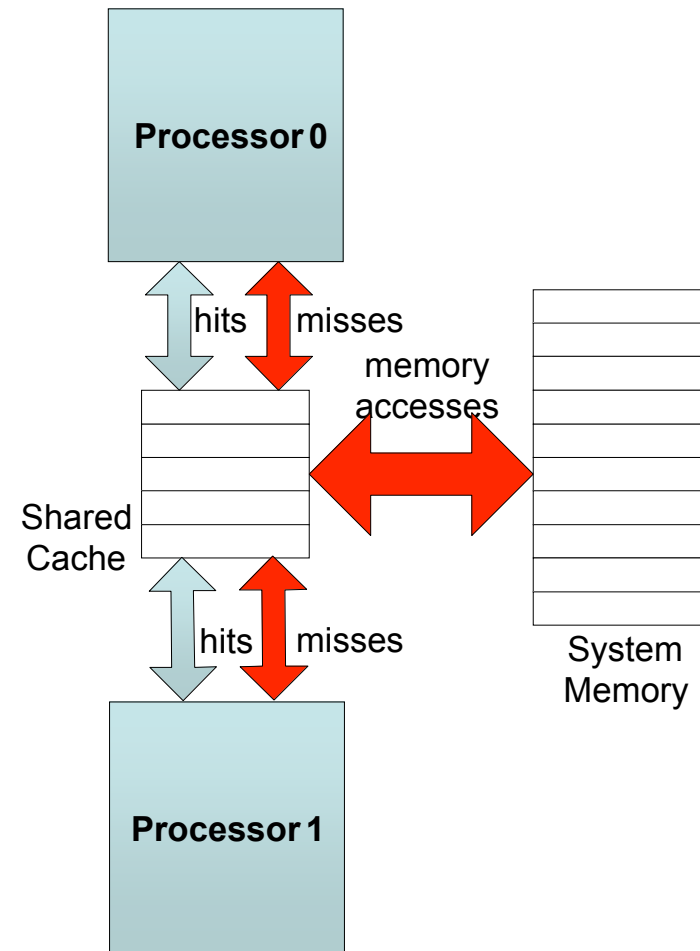
- Chip Multiprocessors (CMPs)
  - Increasingly popular to address the growing demand for higher performance
  - Enable concurrent execution of multiple threads
- No explicit synchronization and communication support for multithreaded applications running on CMPs

# Problems

- Synchronization Overhead
  - Spin Waits
- Memory Bandwidth Bottleneck
  - Many Simultaneous Accesses
- Cache Pollution
  - Data Evictions from Shared Cache
- Demand-Based Data Transfers
  - Depend on Coherence Mechanisms

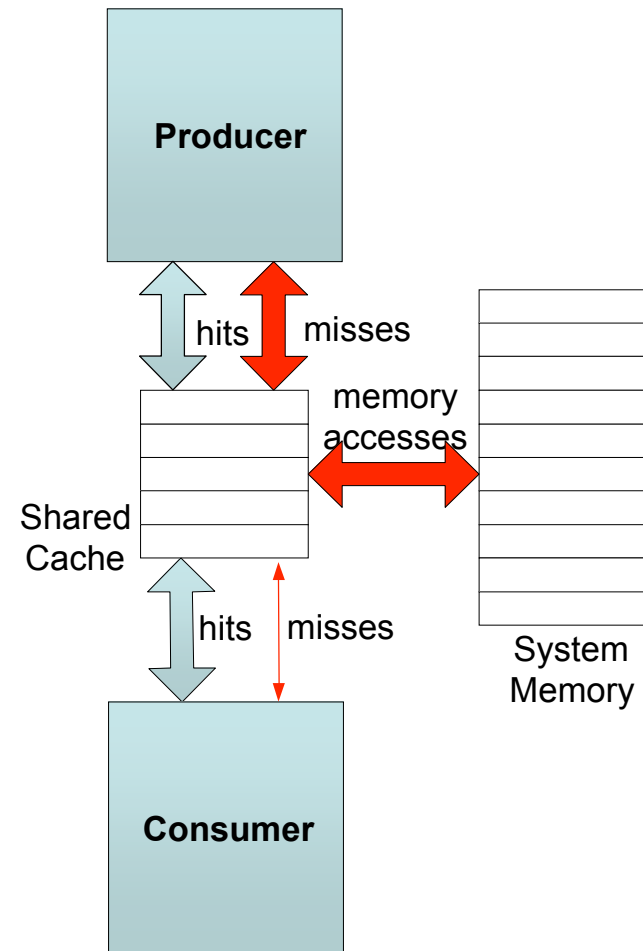
# Conventional Parallel Programming

- Data parallelism by splitting data across multiple threads
- Memory interface is overburdened
- Performance degrades due to large number of cache misses



# Synchronized Pipelined Parallelism Model

- SPPM: Producer-consumer parallelism targeted for CMPs
- Producers and consumers communicate via caches
  - Producer fetches data into cache and modifies it
  - Consumer uses modified data
- A large number of cache misses are converted into hits



# Outline

- Introduction
- **Register-Based Synchronization**
- Data Communications via Prepushing
- Simulation Environment
- Simulation Results
- Conclusion

# Basic Consumer Implementation

```
for i ← 0 to N do
    while sync window for dataBlock[i] is violated
        wait
    read dataBlock[i]
    generate results
```

- No Useful Work During Spin Waits
- Synchronization Window
  - Access synchronization variables in memory
- Coherence Traffic



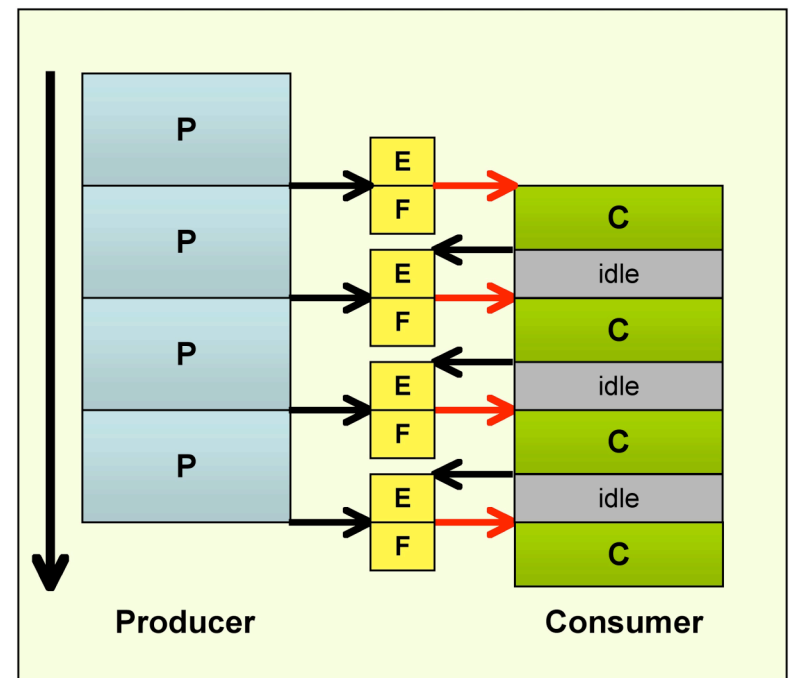
# Basic Consumer Implementation

```
for i ← 0 to N do
    while sync window for dataBlock[i] is violated
        wait
    read dataBlock[i]
    generate results
```

- Optimal Miss Rate: 15%
- L1 Cache Latency: 1 cycle
- L2 Cache Latency: 12 cycles
- Average Access Time
  - = L1 Latency + Miss Rate \* (L1 Latency + L2 Latency)
  - = 3 cycles

# Register-Based Synchronization

- To avoid spin waits in multithreaded applications
- Register-Based Synchronization (RBS)
  - Employs shared registers with full/empty status bits
- Improvements:
  - Reduced miss rates
  - Reduced coherence traffic
  - Reduced execution time
  - Idle mode can save power



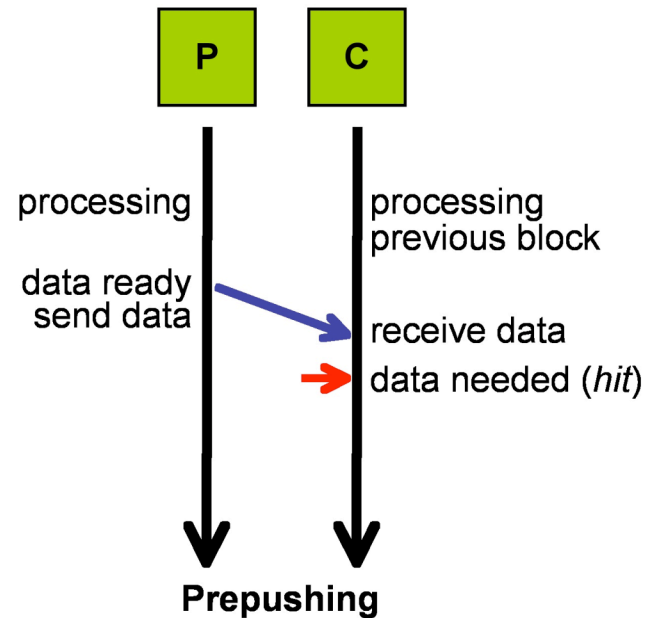
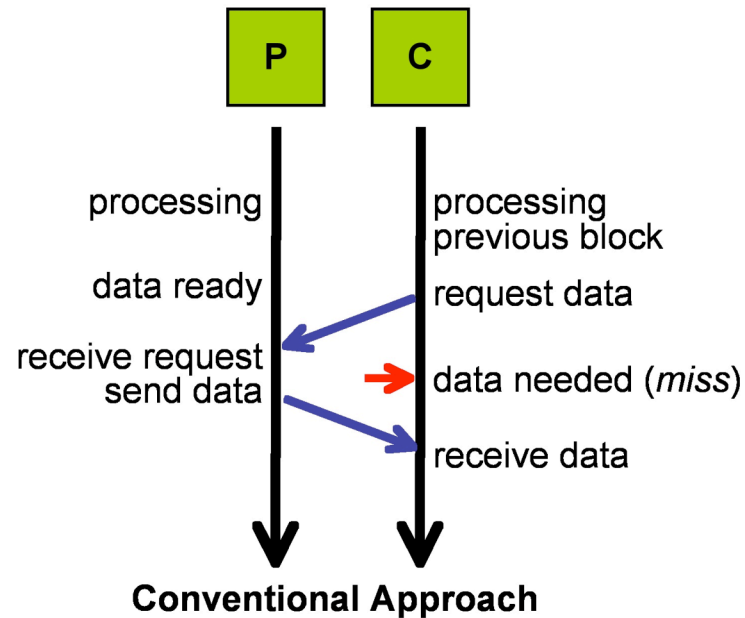
# RBS Implementation

- Memory mapped locations to keep track of synchronization variable accesses
- Device driver allocates kernel memory and allows it to be mapped to user space
- Device's reserved locations accessed as if they were hardware registers

# Outline

- Introduction
- Register-Based Synchronization
- **Data Communications via Prepushing**
- Simulation Environment
- Simulation Results
- Conclusion

# Prepushing

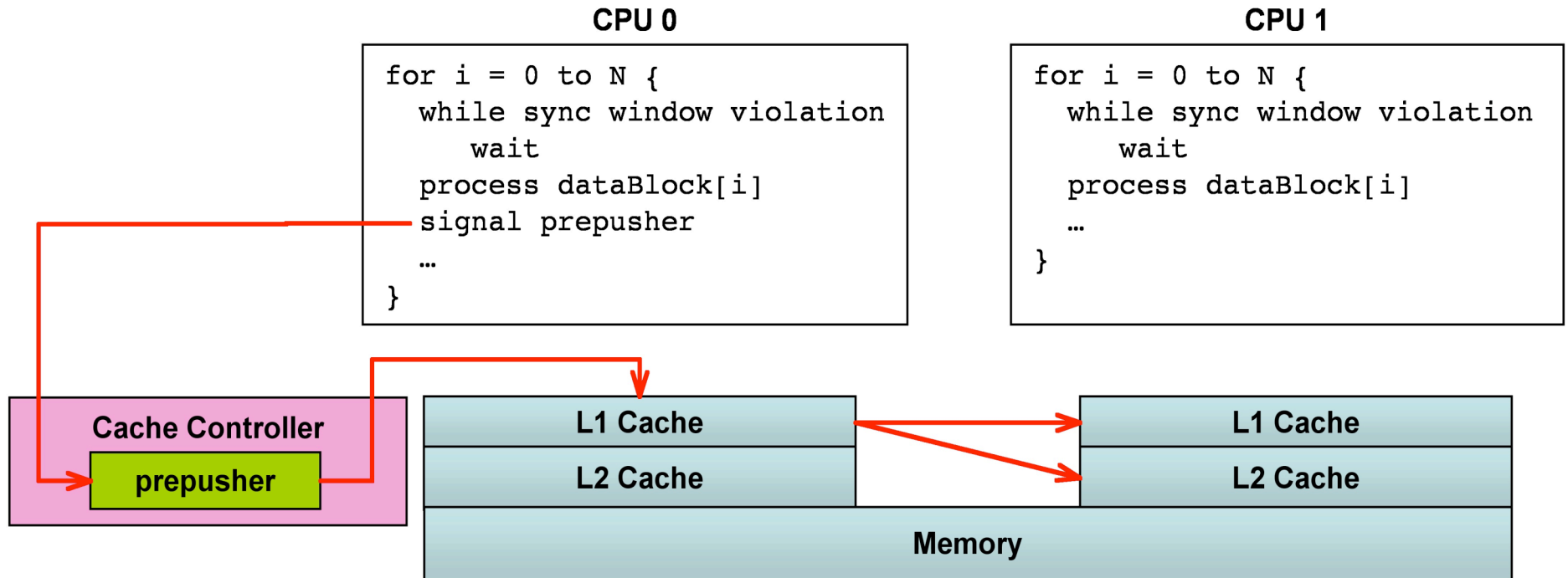


- **Improvements:**
  - Reduced data requests
  - Reduced cache misses
  - Reduced communication latency

# Prepushing Models

- Shared Prepushing
  - PUSH-S-L1
    - send data in shared state, write it to L1 cache
  - PUSH-S-L2
    - send data in shared state, write it to L2 cache
- Exclusive Prepushing
  - PUSH-X-L1
    - send data in exclusive state, write it to L1 cache
  - PUSH-X-L2
    - send data in exclusive state, write it to L2 cache

# Prepushing Implementation



# Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepushing
- **Simulation Environment**
- Simulation Results
- Conclusion



# Benchmarks

- Red-Black Solver
  - Solves a partial differential equation using a finite differencing method
- Finite-Difference Time-Domain Method
  - Extremely memory intensive electromagnetic simulation
- ARC4 Stream Cipher
  - Stream cipher used in protocols such as SSL and WEP

# Simulation Environment

- Simics - Full System Simulator
- GEMS Ruby - Memory Model
- Multi-Core System
  - 2 GHz UltraSPARC III+ Processors
  - L1 Cache: 64 KB, 2-way associative, 1 cycle
  - L2 Cache: 1 MB, 8-way associative, 12 cycles
  - Cache Line Size: 64 B
  - Main Memory: 4 GB, 120 cycles
  - Operating System: Solaris 9

# Outline

- Introduction
- Register-Based Synchronization
- Data Communications via Prepushing
- Simulation Environment
- **Simulation Results**
- Conclusion

# RBS Results

- Estimated Access Time per Iteration  
= Average Access Time \*  
Access Count per Iteration
- RBS Gain  
= Estimated Access Time per Iteration /  
Execution Time per Iteration

# RBS Results (cont'd.)

- RB Solver - RBS Gain: 2-5% per iteration

Grid Size	200 x 200	400 x 400
Exec. Time per Iter. (cycles)	2,056,725	13,905,891
Access Count per Iter.	13,531	211,933
Est. Access Time per Iter.	40,593	635,799

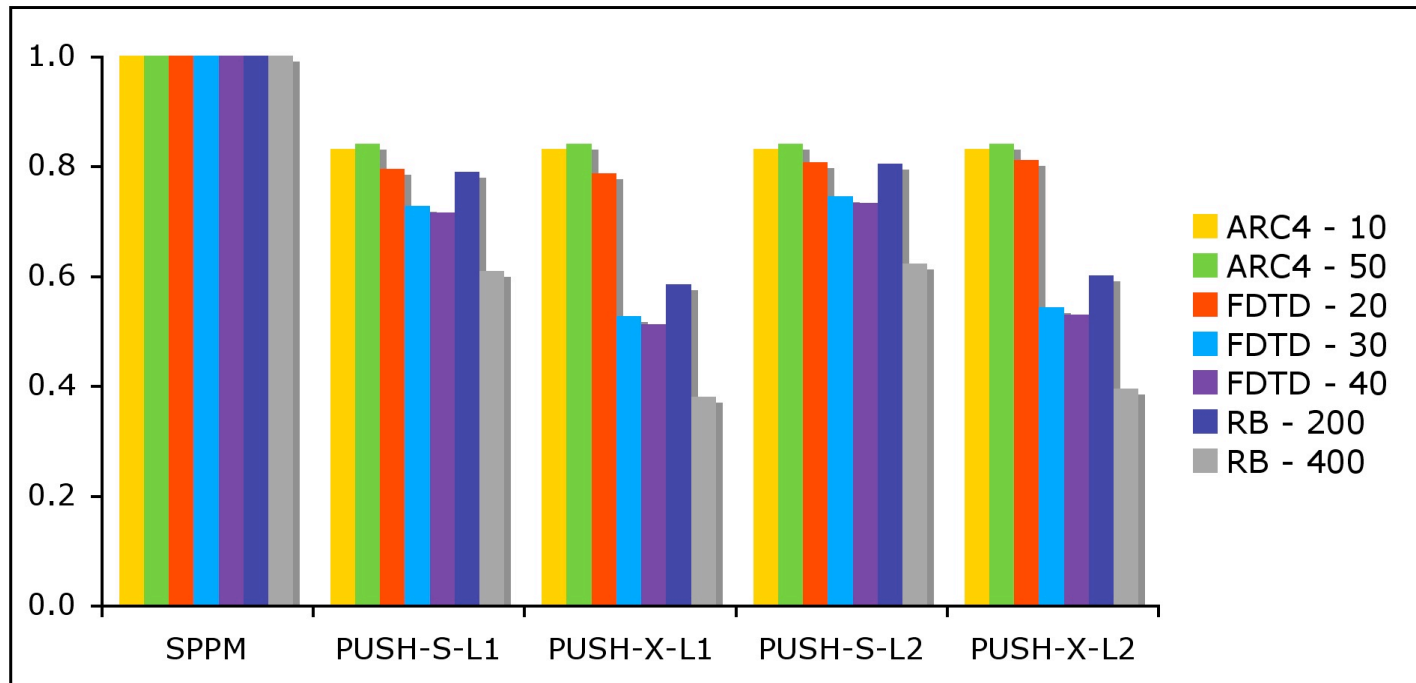
- FDTD - RBS Gain: 6-11% per iteration

Grid Size	30x30x30	40x40x40
Exec. Time per Iter. (cycles)	5,741,480	8,124,822
Access Count per Iter.	120,425	291,506
Est. Access Time per Iter.	361,275	874,518

- ARC4 - RBS Gain: negligible

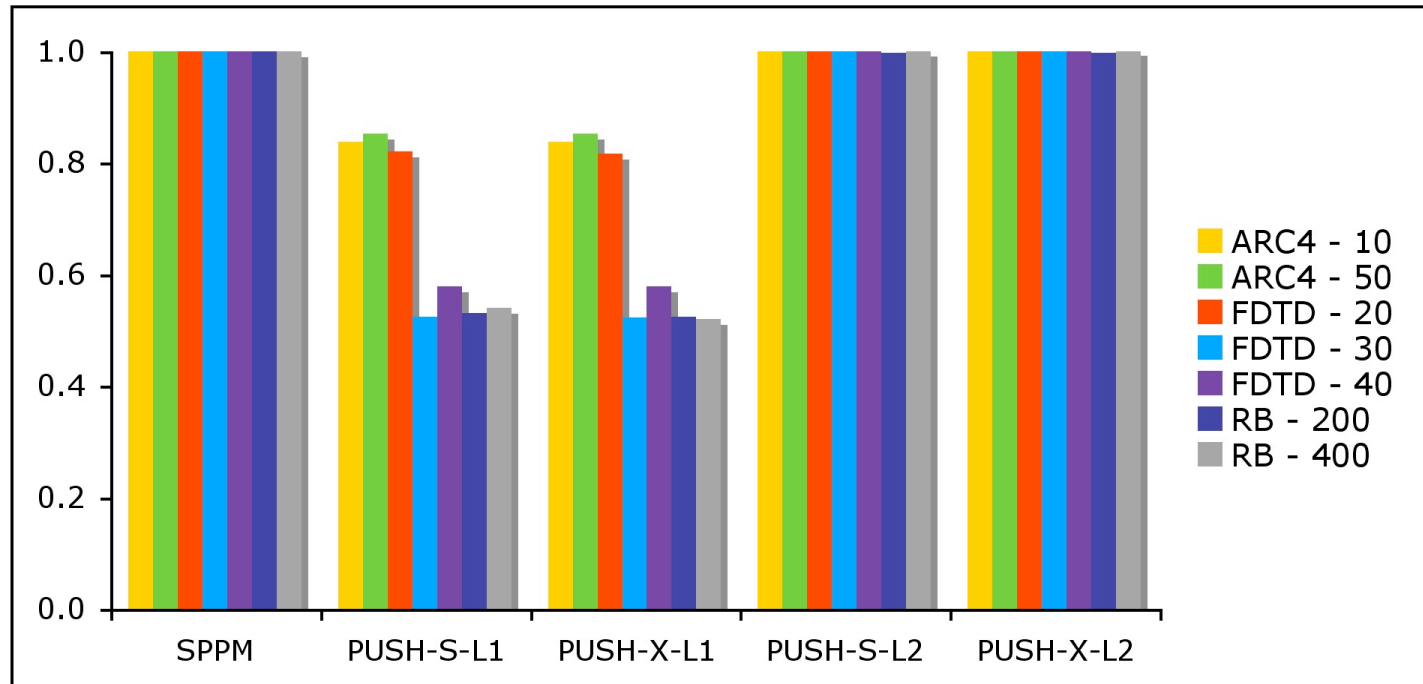
Stream Size	10 MB	50 MB
Exec. Time per Iter. (cycles)	1,327,413	1,330,287
Access Count per Iter.	56	72
Est. Access Time per Iter.	168	216

# Normalized Execution Time



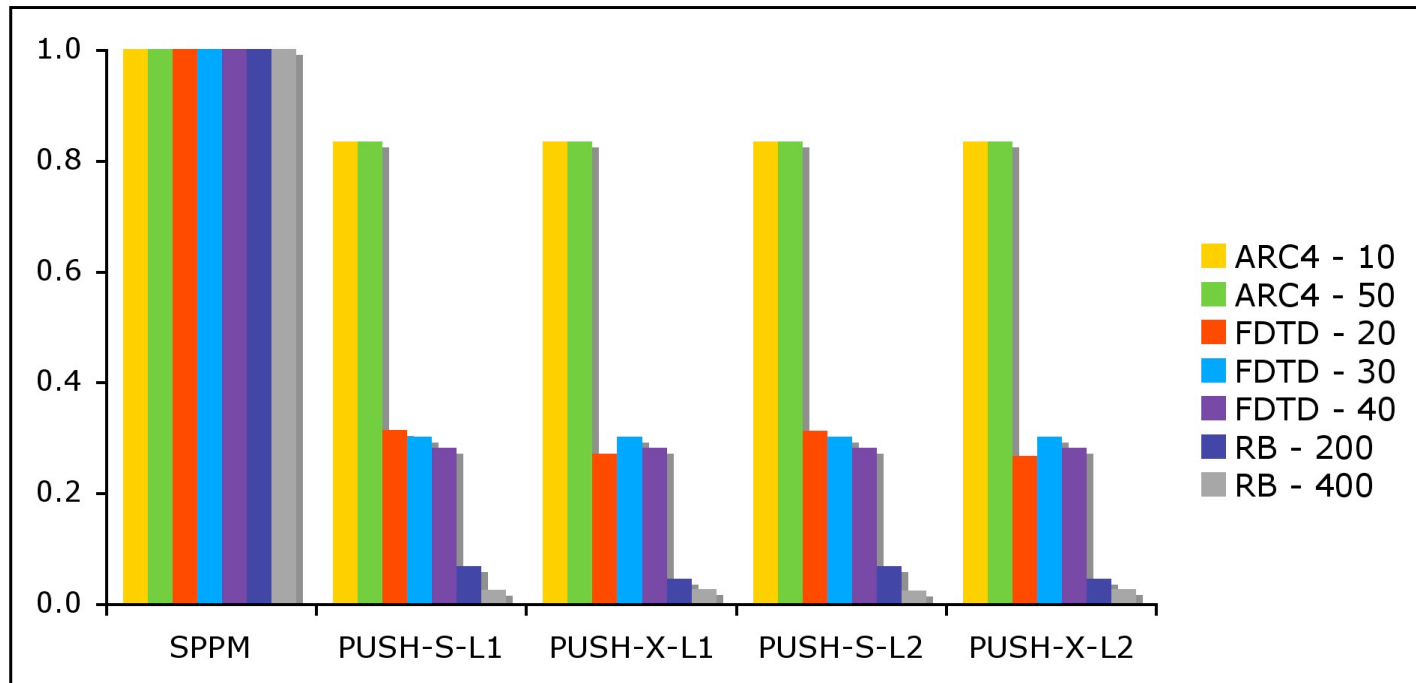
- Improvement depends on application behavior
- Exclusive prepushing models are more effective at reducing execution time

# Consumer's L1D Cache Misses



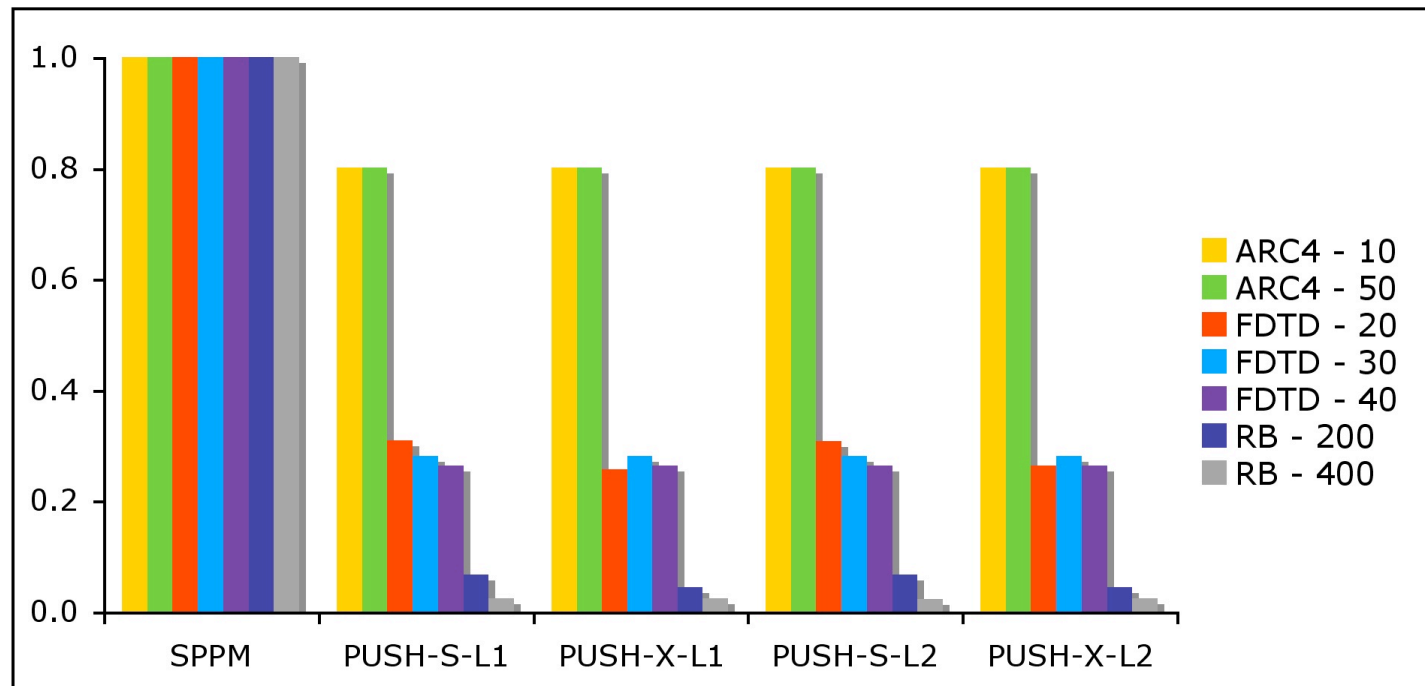
- No improvement in PUSH-S-L2 and PUSH-X-L2 because consumer cannot find its data in L1 cache
- Better than accessing remote cache or main memory

# Consumer's L2 Cache Misses



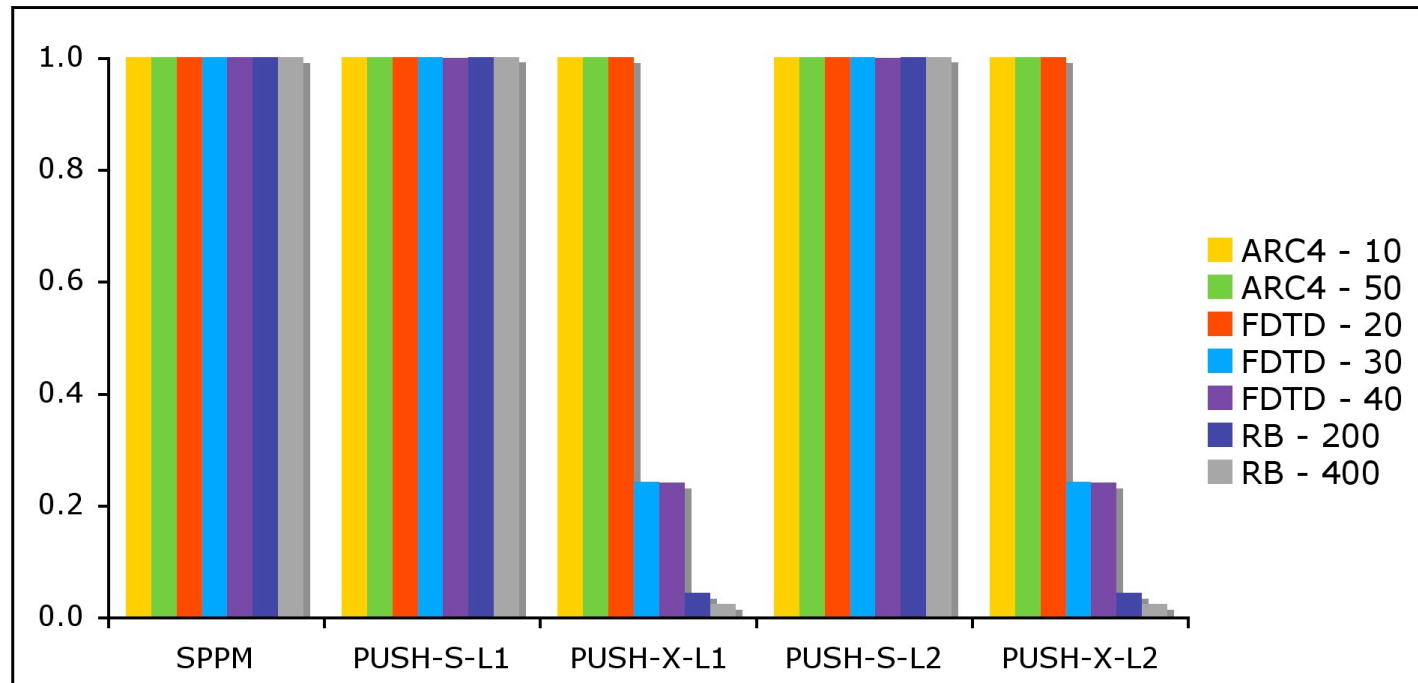


# Consumer's Shared Data Requests



- Not many explicit shared data requests because consumer receives data beforehand

# Consumer's Exclusive Data Requests



- No need to invalidate producer's copy in exclusive prepushing models

# Conclusion

- RBS and Prepushing improve synchronization and communication support for multithreaded applications.
- RBS employs hardware registers to reduce miss rates and help power savings.
- Prepushing provides an efficient communications interface where data can be moved/copied from one cache to another before it is needed at the destination.