Large-Scale Multiprocessors And Scientific Applications

Zhou Li

Chao Sun

Contents

Introduction

- The primary application of large-scale multiprocessors is for true parallel programming
- The primary target of parallel computing is scientific and technical applications

Contents

Interprocessor Communication: The Critical Performance Issue

- Communication bandwidth
- Communication latency

  Communication latency = Sender overhead + Time of flight + Transmission time + Receiver overhead
- Communication latency hiding

Contents

Characteristics

- size of the data items
- regularity in the communication patterns

The FFT Kernel

1. Transpose data matrix.
2. Perform 1D FFT on each row of data matrix.
3. Multiply the roots of unity matrix by the data matrix and write the result in the data matrix.
4. Transpose data matrix.
5. Perform 1D FFT on each row of data matrix.
6. Transpose data matrix.

The LU Kernel

- the blocks of the matrix are assigned to processors using a 2D tiling: $\frac{n}{B} \times \frac{n}{B}$
- the dense matrix multiplication is performed by the processor that owns the *destination* block.

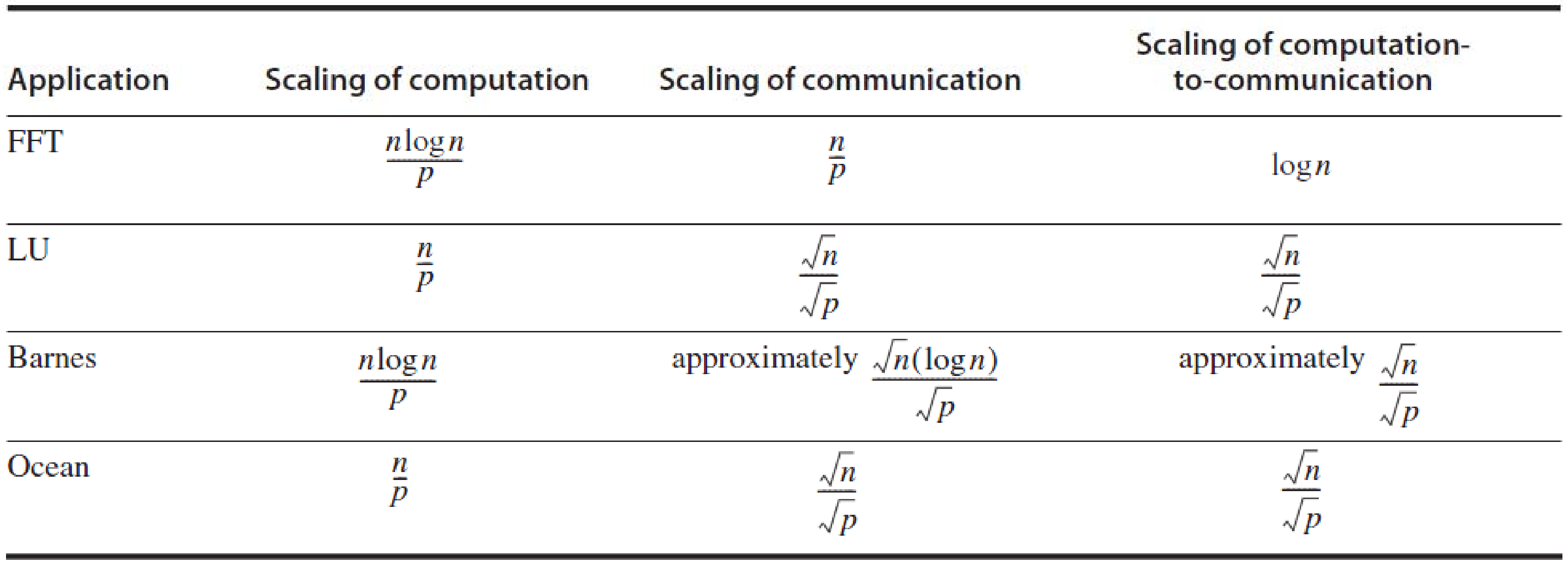
The Barnes Application

- *n*-body algorithm solving a problem in galaxy evolution.
- The Barnes-Hut algorithm uses an octree (each node has up to eight children) to represent the eight cubes in a portion of space

The Ocean Application

- Ocean simulates the influence of eddy and boundary currents on large-scale flow in the ocean
- *Red-black Gauss-Seidel* colors the points in the grid so as to consistently update each point based on previous values of the adjacent neighbors

Computation/Communication Ratio

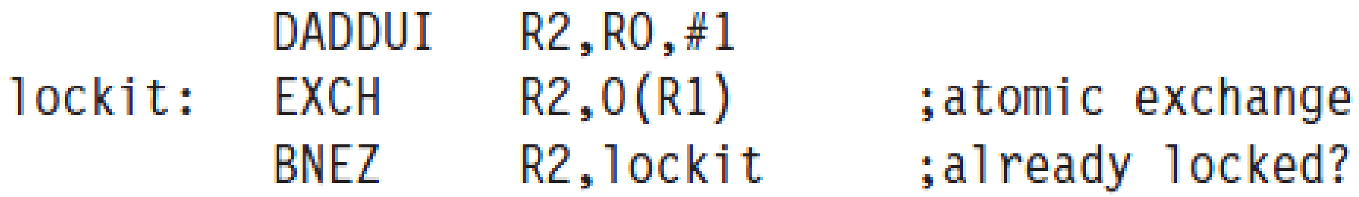


Application	Scaling of computation	Scaling of communication	Scaling of computation-to-communication
FFT	$\frac{n\log n}{p}$	$\frac{n}{p}$	$\log n$
LU	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$
Barnes	$\frac{n\log n}{p}$	approximately $\frac{\sqrt{n}(\log n)}{\sqrt{p}}$	approximately $\frac{\sqrt{n}}{\sqrt{p}}$
Ocean	$\frac{n}{p}$	$\frac{\sqrt{n}}{\sqrt{p}}$	$\frac{\sqrt{n}}{\sqrt{p}}$

Contents

Synchronization: Spinning Lock

- locks that a processor continuously tries to acquire, spinning around a loop until it succeeds



```
        DADDUI   R2,R0,#1
lockit: EXCH     R2,0(R1)      ;atomic exchange
        BNEZ     R2,lockit     ;already locked?
```

Barrier Synchronization

```
lock (counterlock);/* ensure update atomic */
if (count==0) release=0;/* first=>reset release */
count = count + 1;/* count arrivals */
unlock(counterlock);/* release lock */
if (count==total) {/* all arrived */
       count=0;/* reset counter */
       release=1;/* release processes */
}
else {/* more to come */

       spin (release==1);/* wait for arrivals */
}
```

Synchronization Performance Challenges

- Contention
  - synchronization performance can be a real bottleneck when there is substantial contention among multiple processes.
- Serialization
  - This serialization is a problem when there is contention because it greatly increases the time to complete the synchronization operation.

Exponential Back-off And Queuing Locks

```
         DADDUI    R3,R0,#1     ;R3 = initial delay
lockit:  LL        R2,0(R1)     ;load linked
         BNEZ      R2,lockit    ;not available-spin
         DADDUI    R2,R2,#1     ;get locked value
         SC        R2,0(R1)     ;store conditional
         BNEZ      R2,gotit     ;branch if store succeeds
         DSLL      R3,R3,#1     ;increase delay by factor of 2
         PAUSE     R3           ;delays by value in R3
         J         lockit
gotit:   use data protected by lock
```

Contents

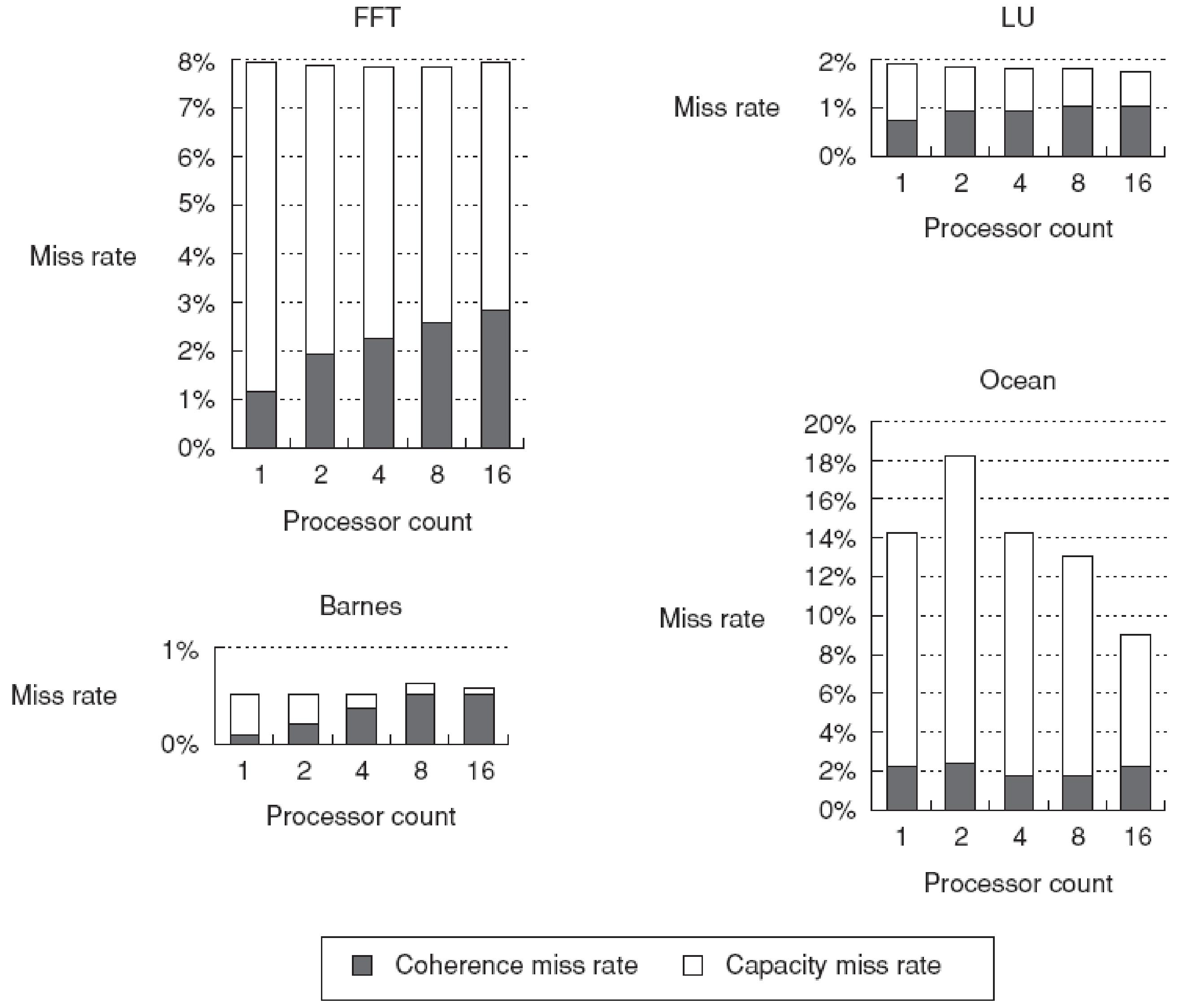
Performance Evaluation on Symmetric Shared-Memory Multiprocessors

- Normal uniprocessor misses VS coherence misses
- Capacity misses dominate uniprocessor misses
- Does not distinguish between private and shared cache block
- Vary processor number, cache size and block size

Symmetric Shared-Memory Multiprocessor Increase The Processor Count

- Increase the total amount of cache, causing capacity misses to drop
- Increase the amount of communication, causing coherence misses to rise
- Some application has extraordinary performance change due to its specific characteristics

Symmetric Shared-Memory Multiprocessor Miss Rate VS Processor Count



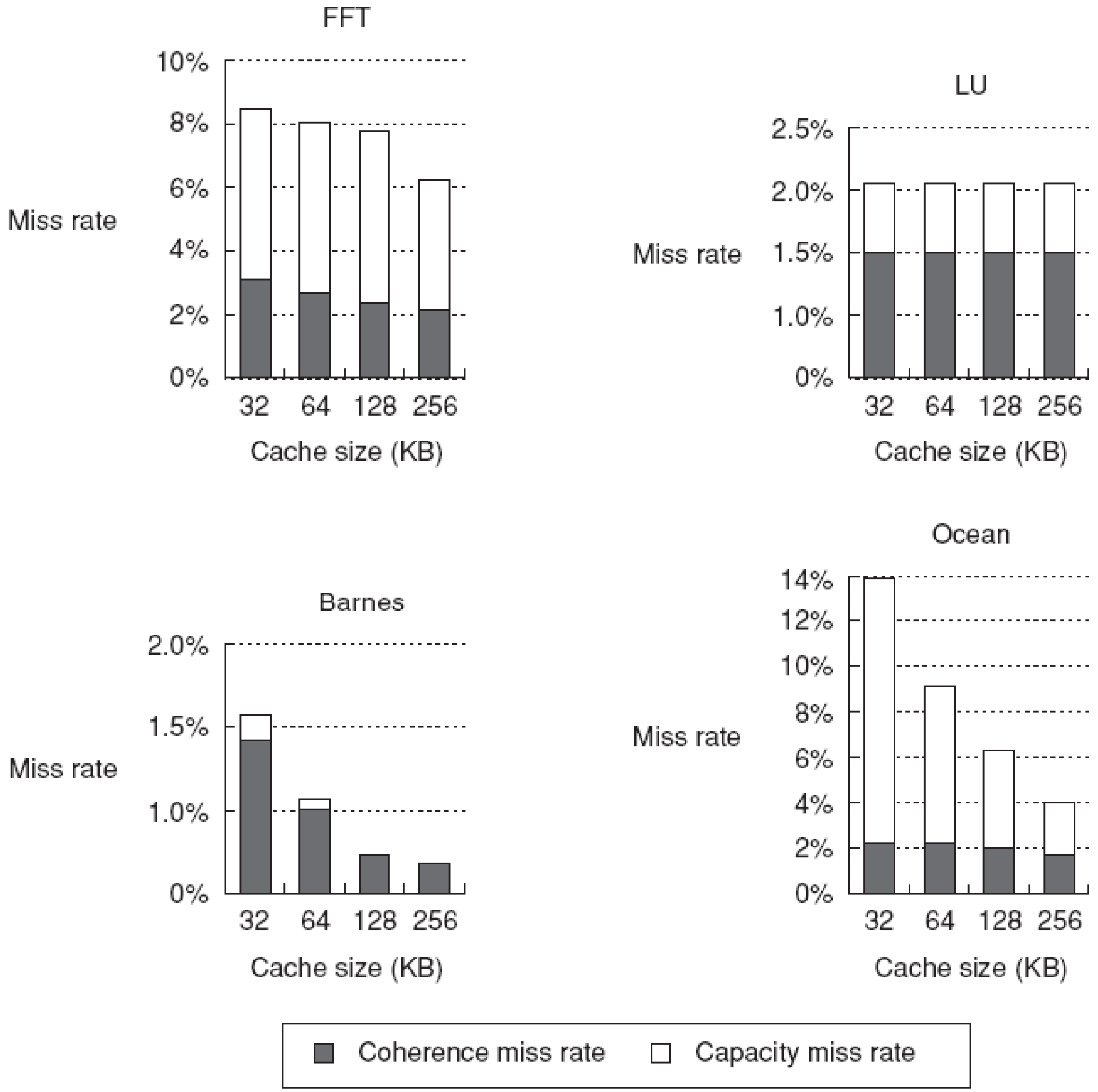
Symmetric Shared-Memory Multiprocessor Increase The Cache Size

- Usually has a beneficial effect as it reduces cache misses
- Two reasons for non-obvious reduction of miss rates

  - Inherent communication

  - Temporary plateau

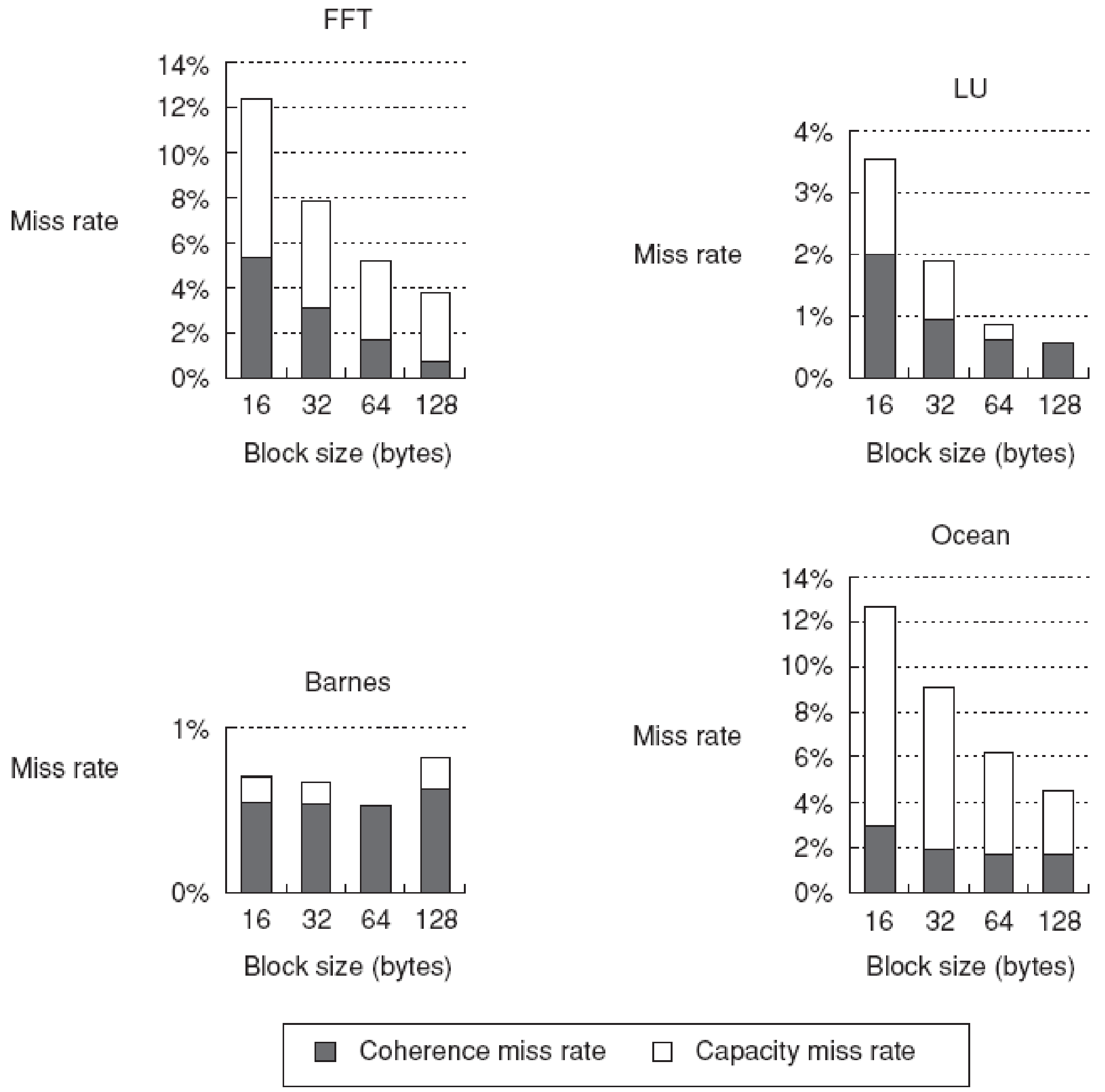
Symmetric Shared-Memory Multiprocessor Miss Rate VS Cache Size



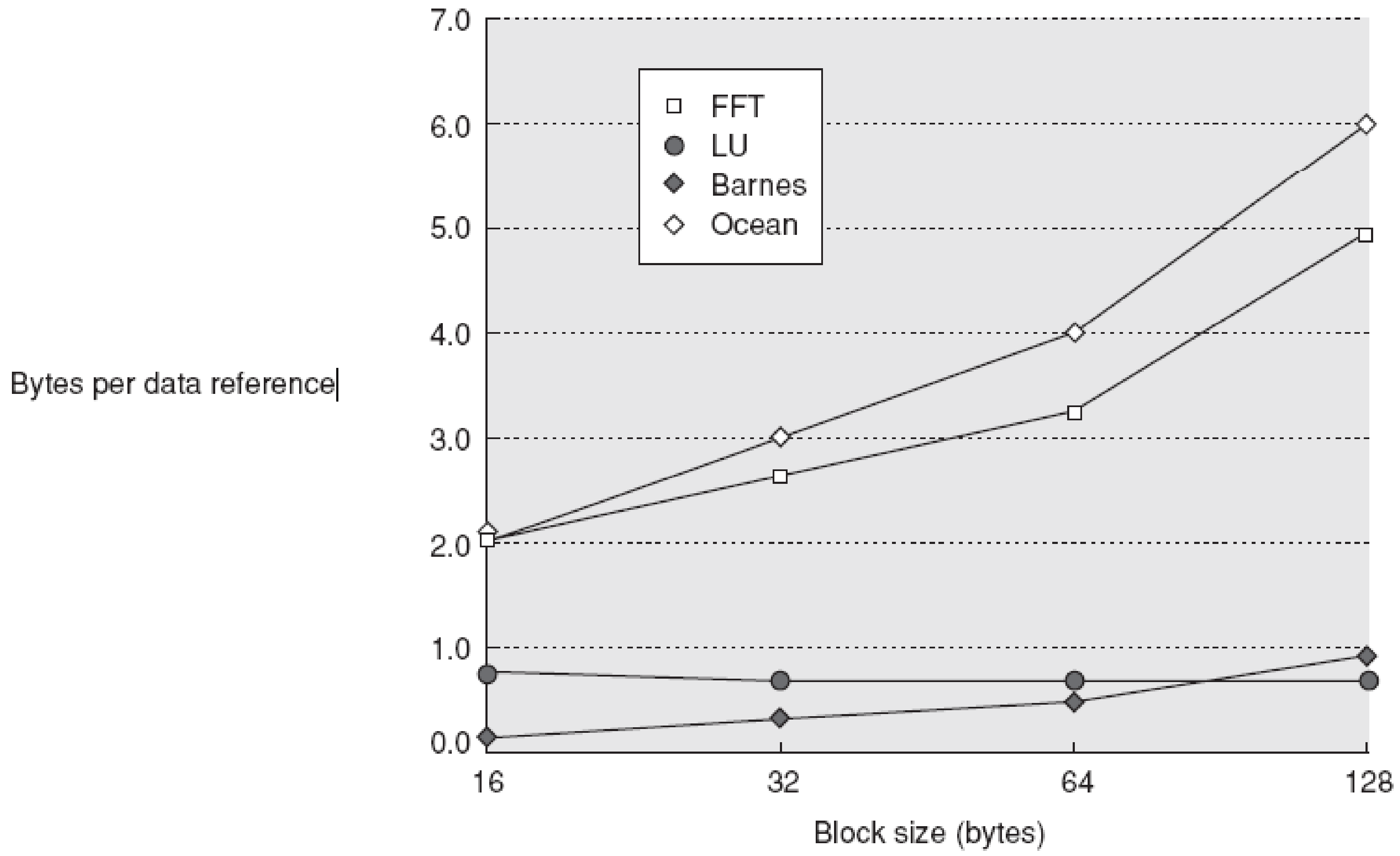
Symmetric Shared-Memory Multiprocessor Increase The Block Size

- Reduces spatial locality for shared data
- Potential increase in miss rate due to false sharing
- Real bottle-neck in bus-based multiprocessors is the limited memory and bus bandwidth
- The growth in traffic can actually lead to performance slowdowns due to longer miss penalties and increased bus contentions

Symmetric Shared-Memory Multiprocessor Miss Rate VS Block Size



Symmetric Shared-Memory Multiprocessor Bytes per Data Reference VS Block Size



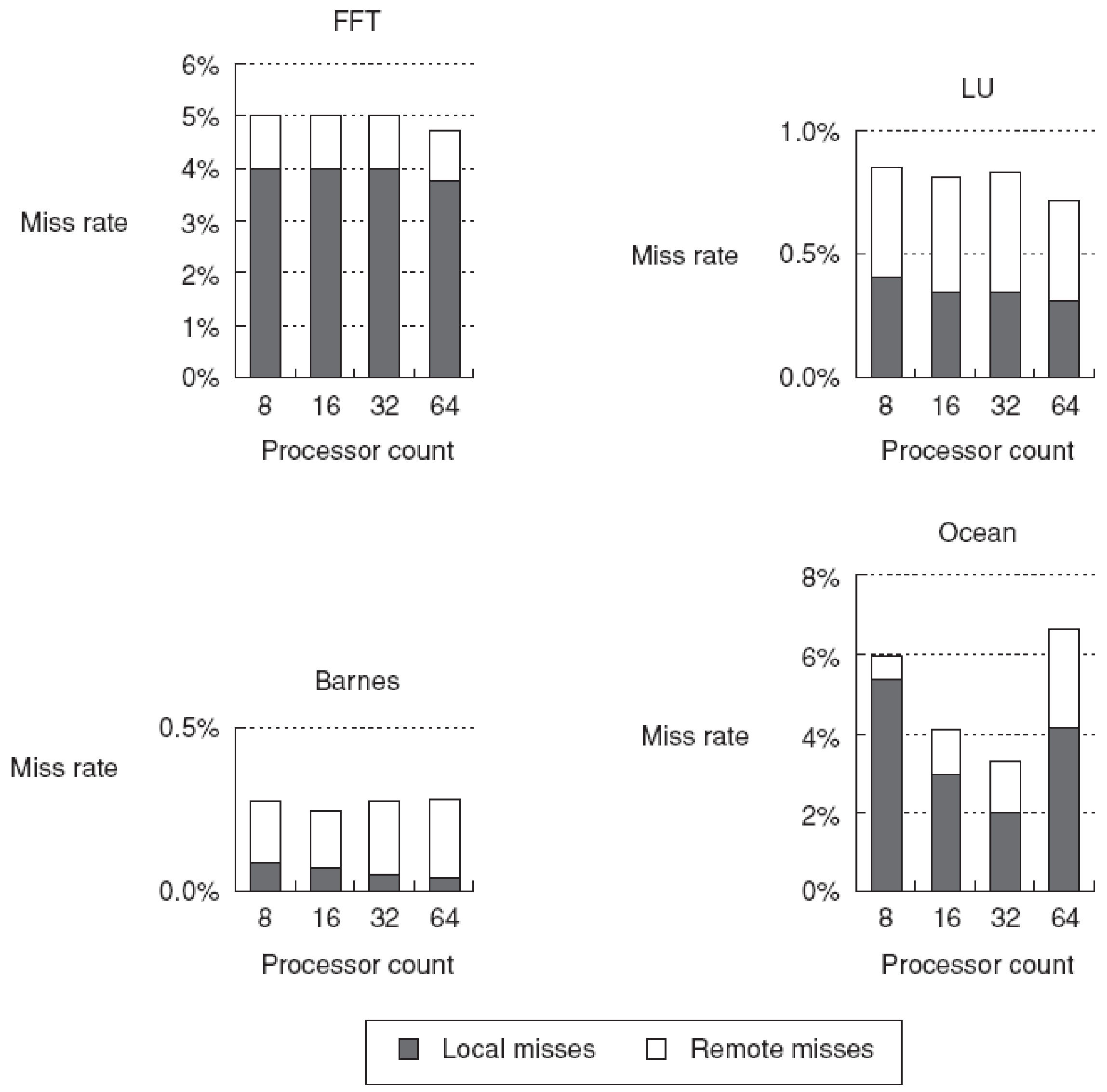
Performance Evaluation on Distributed-Memory Multiprocessors

- The distribution of memory requests between local and remote is key to performance, as it affects global bandwidth consumption and request latency
- Separate the cache misses into local and remote requests
- Coherence misses dominate remote misses

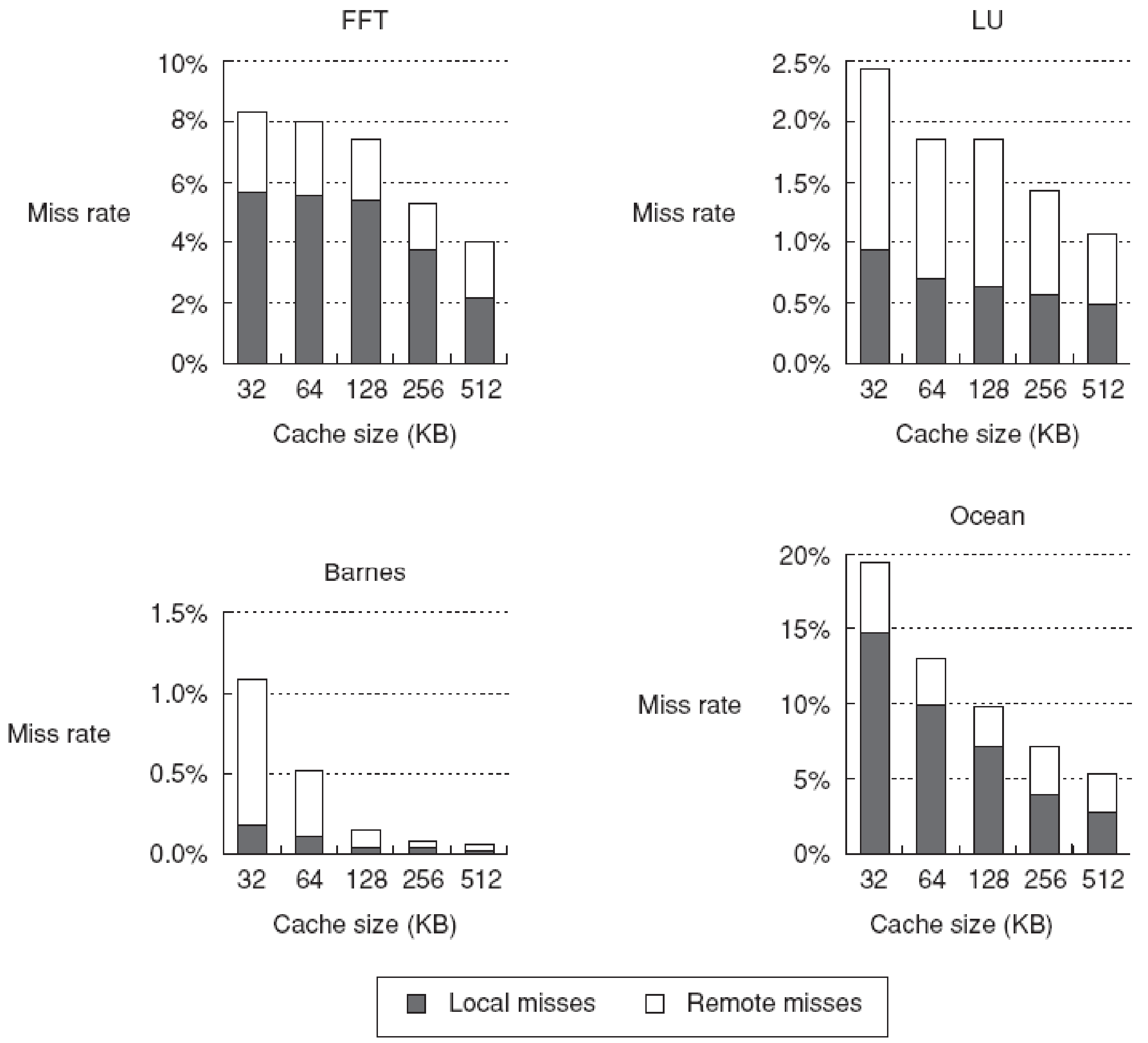
Performance Changes on Distributed-Memory Multiprocessors

- Effect of processor count increase is little
- Miss rates decrease as cache sizes grow
- Miss rates drop as block sizes grow
  - good spatial locality of sample applications
  - local misses reduction plays the key role
  - still need to consider the local and global bandwidth consumptions
- Extraordinary changes due to specific characteristics

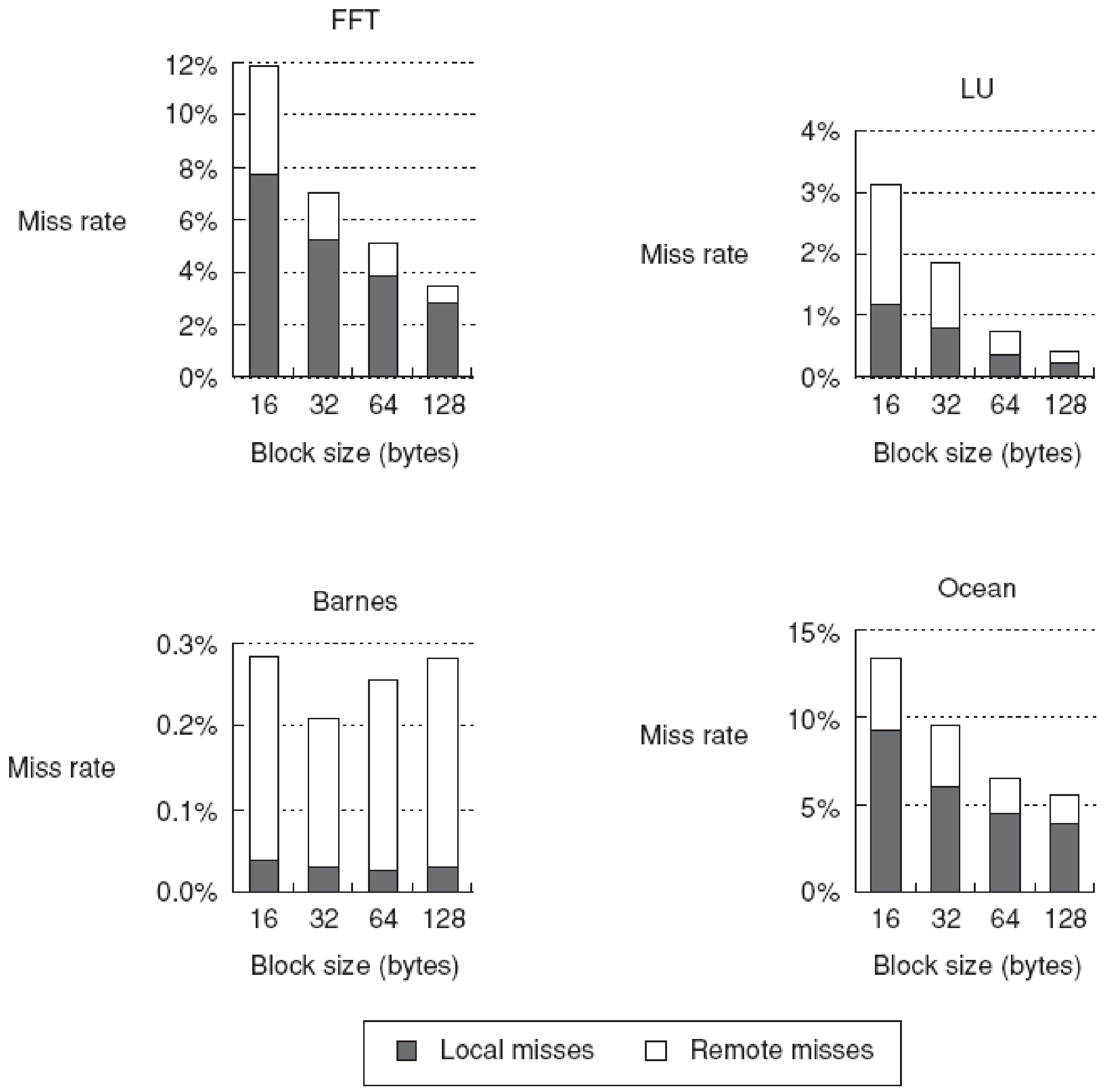
Distributed-Memory Multiprocessor Miss Rate VS Processor Count



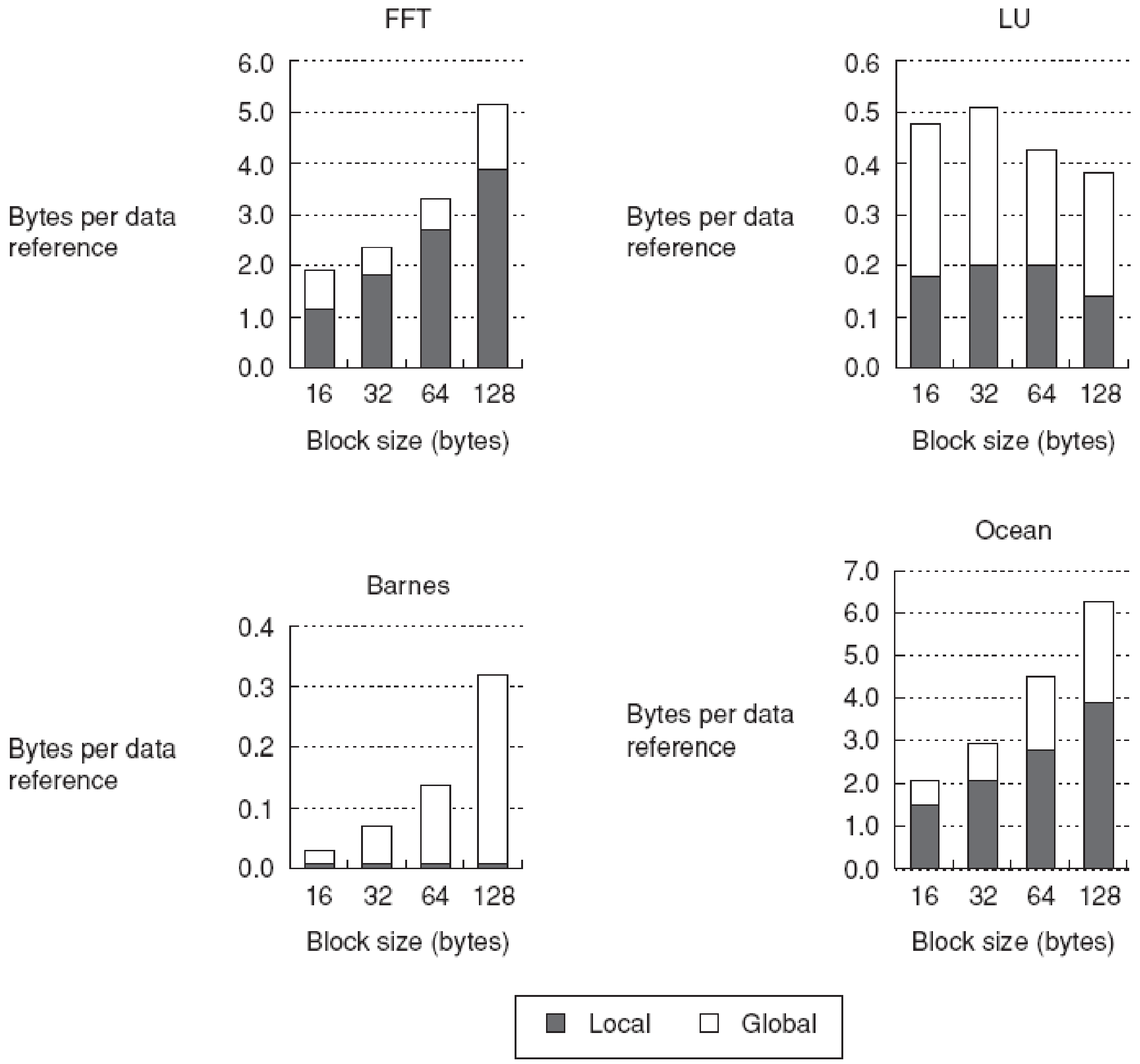
Distributed-Memory Multiprocessor Miss Rate VS Cache Size



Distributed-Memory Multiprocessor Miss Rate VS Block Size

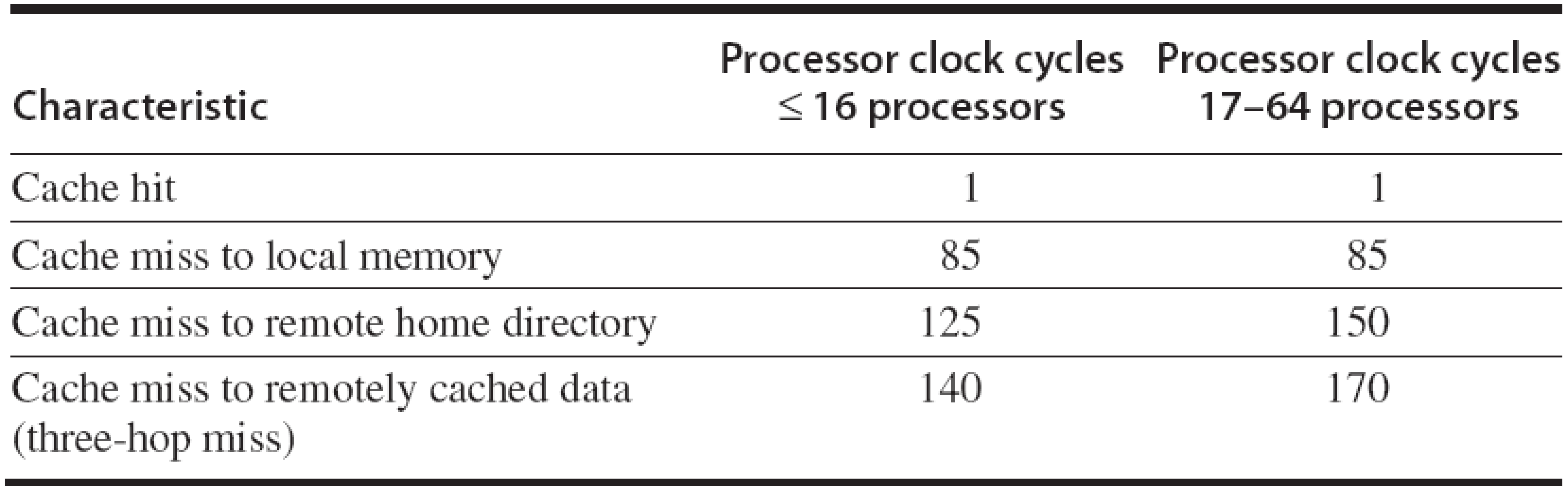


Distributed-Memory Multiprocessor Bytes per Data Reference VS Block Size



Distributed-Memory Multiprocessor Remote Memory Access Latency

- Also a key issue for the performance
- Contention may have a serious impact



Characteristic	Processor clock cycles ≤ 16 processors	Processor clock cycles 17–64 processors
Cache hit	1	1
Cache miss to local memory	85	85
Cache miss to remote home directory	125	150
Cache miss to remotely cached data (three-hop miss)	140	170

Distributed-Memory Multiprocessor Average Memory Reference Cost (cycles)

- Influenced by total frequency of cache misses and miss location distribution
- Need to consider the effect of contentions and synchronization delays

Contents

Performance Measurement of Parallel Processors

- Measuring only CPU time may be misleading
- Scaling the application is more interesting

  - un-scaled version may give pessimistic result

  - select uniprocessor measuring algorithm

  - two ways of scaling

    * memory-constrained scaling, keep memory per processor constant

    * time-constrained scaling, keep total execution time constant, need to know the relationship between running time and problem size
- Scaling may affect result quality

Contents

Implementing Cache Coherence for Snoopy Protocol

- Update miss handling needs multiple steps
  - compose an invalidate message
  - transmit message when network is available
  - process request after all invalidates have been processed
- Challenge is to make the process appear atomic
  - how to resolve the race?
  - how to know all invalidates have been processed?
- Solutions for the difficulties
  - requests pass through a single channel, communication network accepts message only when delivery is guaranteed
  - race losers invalidate the block and generate a write miss to get data, the winner ignores this invalidate
  - or let incoming requests precede over outgoing ones

Implementing Cache Coherence in A DSM Multiprocessor

- Serializations on competing writes and memory consistency need to be enforced
  - exclusive access request is easy as directory is unique
  - ensure that write is complete before the next is begun
  - directory signals to winner when on completing processing
  - sends to race losers a negative acknowledge (NAK)
- Know when the invalidates are complete
  - the destination node of the invalidate message explicitly acknowledge this message from the directory
  - acknowledgement can be sent to the directory or the requester
  - the latter one reduces the possibility of creating a bottleneck at a directory

Avoiding Deadlock from Limited Buffering

- Deadlock situations
  - multiple resources are needed for completing
  - resources are held until completing
  - no global partial order on acquisition of resources
- Ensure resources will always be available for completing a transaction
  - new requests cannot block replies that free up buffers
  - allocate space for requests to accept reply (if expected)
  - can reply a request with NAK, but never NAK a reply
  - requests retry on receiving NAK
- Ensure all replies can be accepted and all requests will be serviced
  - e.g. all requests could be completed

Implementing The Directory Controller

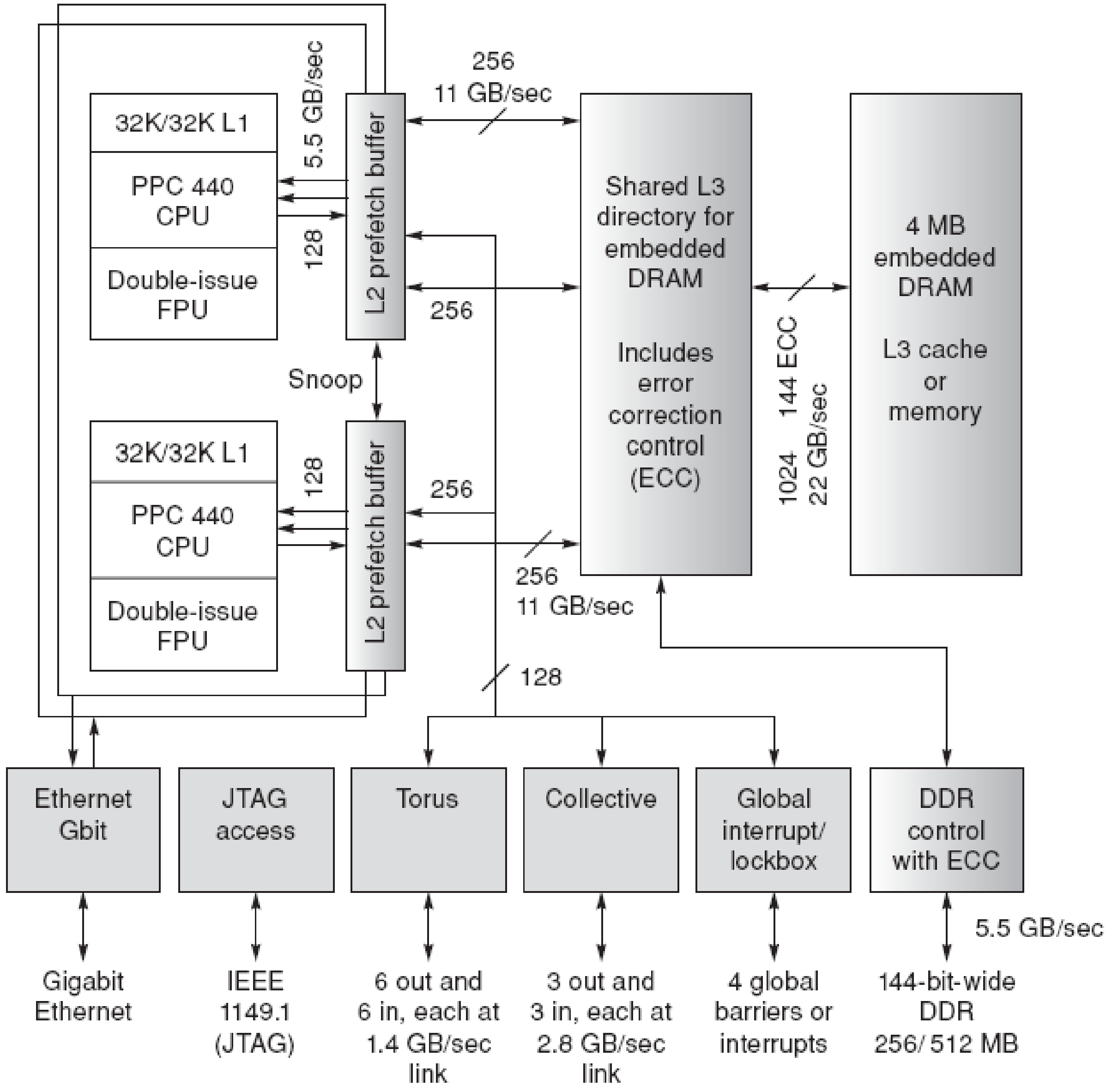
- Ability to NAK requests when receiving too many of them
- Be multithreaded and able to handle requests for multiple blocks independently
- Capability of suspending execution while waiting for replies
- Dealing with non-atomicity and finite buffering is critical

Contents

Blue Gene/L: Parameters And Characteristics

- Distributed-memory, message-passing
- Special customized processing node with two processors, caches, and interconnect logic
- Up to 64K nodes organized into 32 racks
- 85% interconnect within single rack, greatly reducing inter-rack complexity and latency
- All the logic into a single chip, higher density, lower power, and lower cost

Blue Gene/L Processing Node



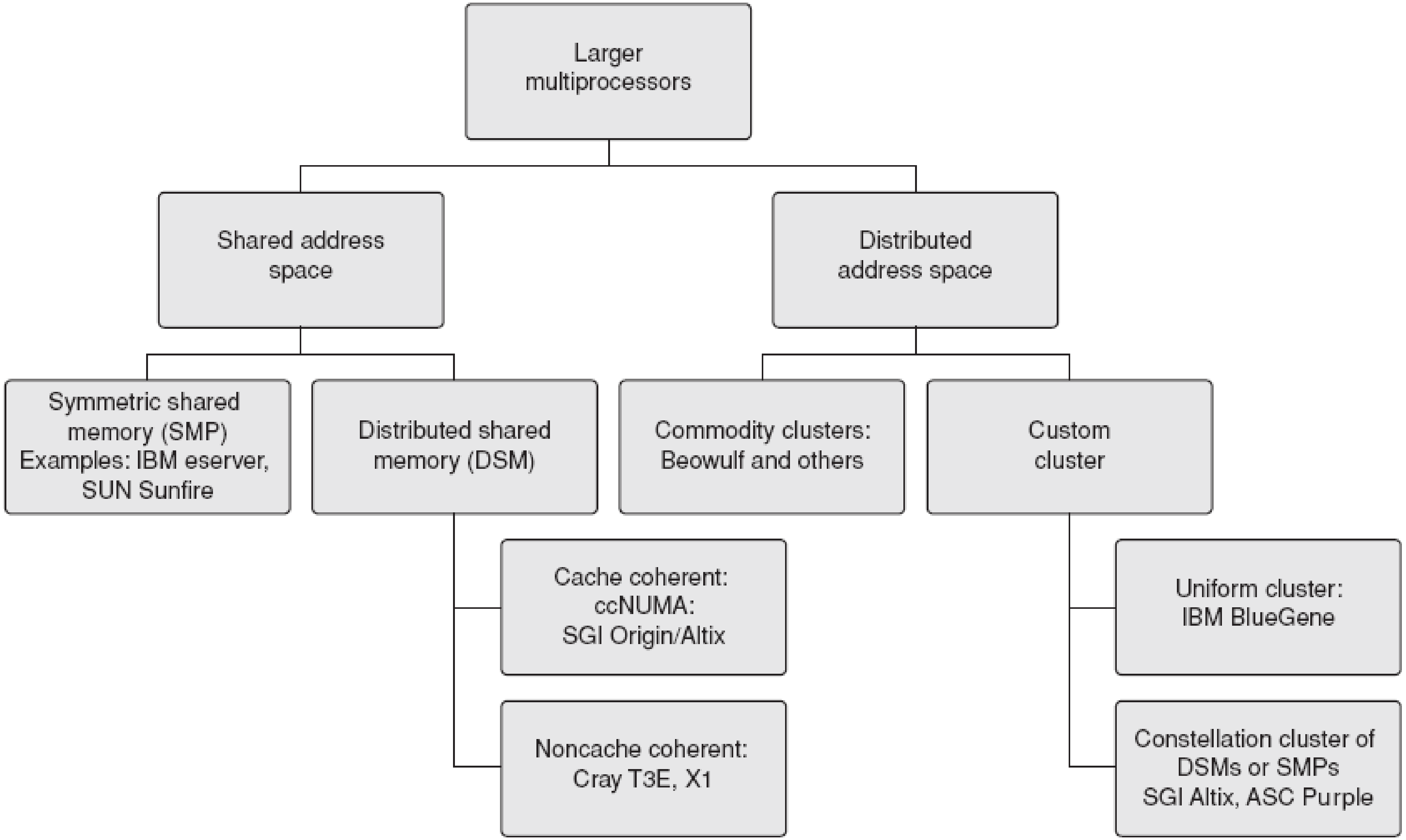
The 64k-processor Blue Gene/L System

Contents

Classification of Large-scale Multiprocessors

Terminology	Characteristics	Examples
MPP	Originally referred to a class of architectures characterized by large numbers of small, typically custom processors and usually using an SIMD style architecture.	Connection Machines CM-2
SMP (symmetric multiprocessor)	Shared-memory multiprocessors with a symmetric relationship to memory; also called UMA (uniform memory access). Scalable versions of these architectures used multistage interconnection networks, typically configured with at most 64–128 processors.	SUN Sunfire, NEC Earth Simulator
DSM (distributed shared memory)	A class of architectures that support scalable shared memory in a distributed fashion. These architectures are available both with and without cache coherence and typically can support hundreds to thousands of processors.	SGI Origin and Altix, Cray T3E, Cray X1, IBM p5 590/5
Cluster	A class of multiprocessors using message passing. The individual nodes are either commodities or customized, likewise the interconnect.	See commodity and custom clusters
Commodity cluster	A class of clusters where the nodes are truly commodities, typically headless workstations, motherboards, or blade servers, connected with a SAN or LAN usually accessible via an I/O bus.	"Beowulf" and other "homemade" clusters
Custom cluster	A cluster architecture where the nodes and the interconnect are customized and more tightly integrated than in a commodity cluster. Also called distributed memory or message passing multiprocessors.	IBM Blue Gene, Cray XT3
Constellation	Large-scale multiprocessors that use clustering of smaller-scale multiprocessors, typically with a DSM or SMP architecture and 32 or more processors.	Larger SGI Origin/ Altix, ASC Purple

Large-scale Multiprocessors Class Hierarchy


Larger multiprocessors
Shared address space
Distributed address space
Symmetric shared memory (SMP) Examples: IBM eserver, SUN Sunfire
Distributed shared memory (DSM)
Commodity clusters: Beowulf and others
Custom cluster
Cache coherent: ccNUMA: SGI Origin/Altix
Noncache coherent: Cray T3E, X1
Uniform cluster: IBM BlueGene
Constellation cluster of DSMs or SMPs SGI Altix, ASC Purple

Emerging Trends from A Look at The TOP500 Multiprocessor List

- Clusters represent a majority of the systems
- The majority of the clusters are commodity clusters, often put together by users
- Although commodity clusters dominate the list, Top 25 are much more varied
- Vector processors, which once dominated the list, have almost disappeared
- IBM Blue Gene dominates the top 10 systems
- Architectural convergence has been driven more by market effects

Thank You