High Performance Computing Systems

- Operating Systems

Doug Shook

Operating Systems - Historical

- First supercomputers were specially designed, few "cores"
  - OSes were mostly job schedulers

- As specialized hardware fell out of use, so did specialized OSes – to a point

Operating Systems – Modern Usage

- A quick glance at TOP500 tells part of the story

- It is clear that Linux is the OS of choice...
  - But why?

Benefits of Linux in HPC

- Modularity
- Generic
- Scalability
- Open Source
- Cost
- Architecture
- Tradition

Operating System Considerations

- Types of nodes
- Communication
- Idea: Segment the OS to match the node types

Distributed OS

- Each node has a microkernel
- System management is kept separate
- From the user's standpoint, it should appear transparent

Distribution Models

- Centralized
- Decentralized
- Fully distributed

A Case Study: IBM BlueGene/L

- Moreira, et. al., “Designing a Highly-Scalable Operating System: The Blue Gene/L Story”

- Blue Gene/L

Blue Gene/L OS Design

- Budgeting / Scheduling
- Avoid pitfalls from previous designs, maintain a scalable solution
- Flexibility
- Main design principle: software structure should reflect hardware structure

Blue Gene/L Architecture

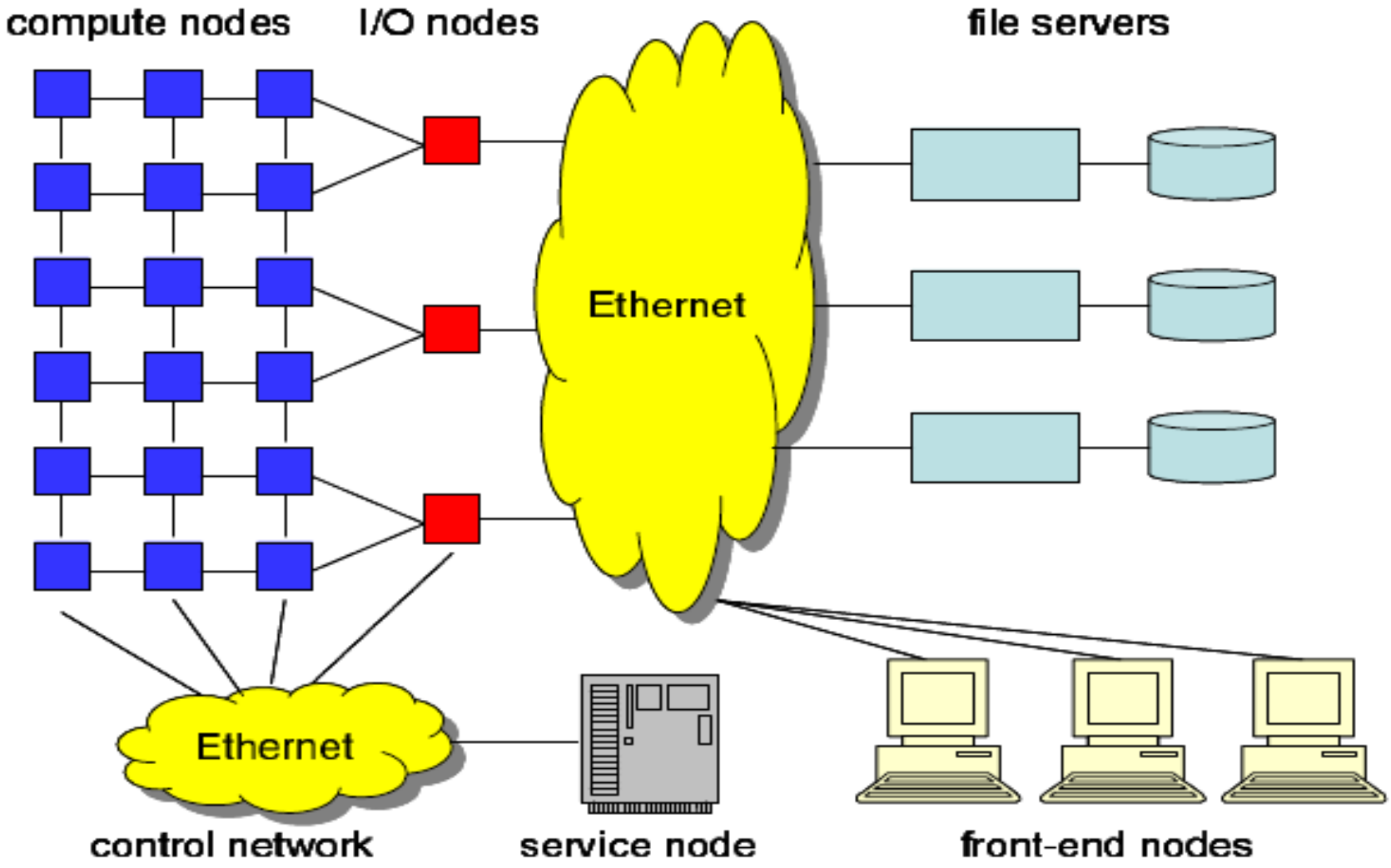


Figure 1: High-Level View of a Blue Gene/L System.

Blue Gene/L Architecture

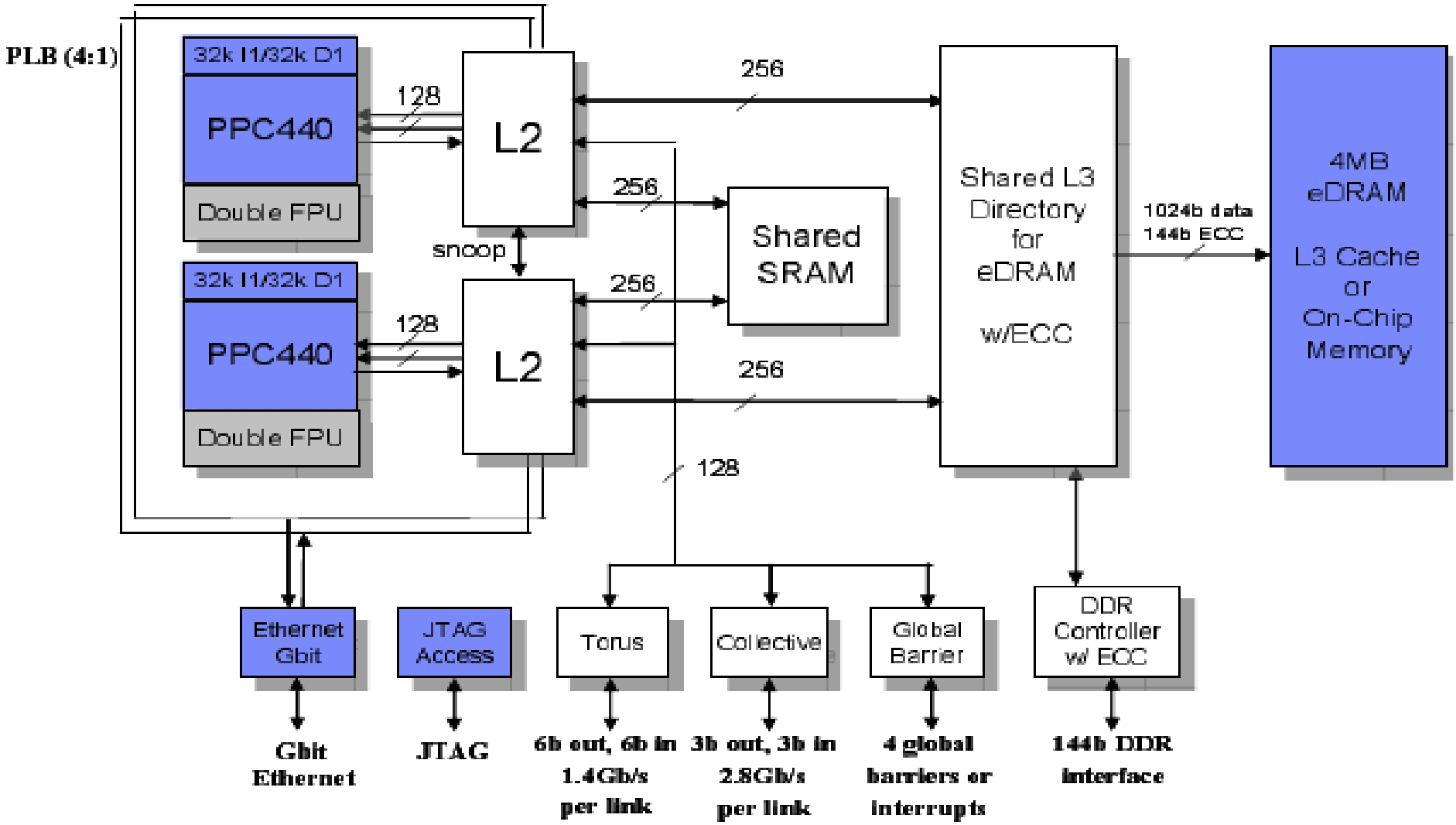


Figure 2: Blue Gene/L Compute ASIC (BLC).

Blue Gene/L OS Architecture

- Compute and I/O nodes are broken into processing sets (psets)
- Node partitions consist of psets
- Psets are implemented purely in software

Blue Gene/L Job Model

- Jobs consist of N processes
- I/O nodes are closer to conventional computers

Compute Node Kernel (CNK)

- Simplified operation
- All for only 1MB of space
- No memory management
- No scheduling

CNK – System Calls

- Simple: Handled normally by the kernel
- I/O: shipped to I/O node of the pset
- Unsupported: return error conditions
- 68 system calls from linux + 18 CNK specific calls

I/O Node operation

- Two roles:
  - Distributing work / data within psets
  - Service requests from compute nodes

I/O Node OS

- Embedded Linux
  - No swap space
  - In-memory root file system
  - Lacks most daemons and services

- Main process: control and I/O daemon (CIOD)
  - Links compute nodes to outside world
  - Passes job parameters, arguments, environment variables to compute nodes

CIOD – Function Shipping

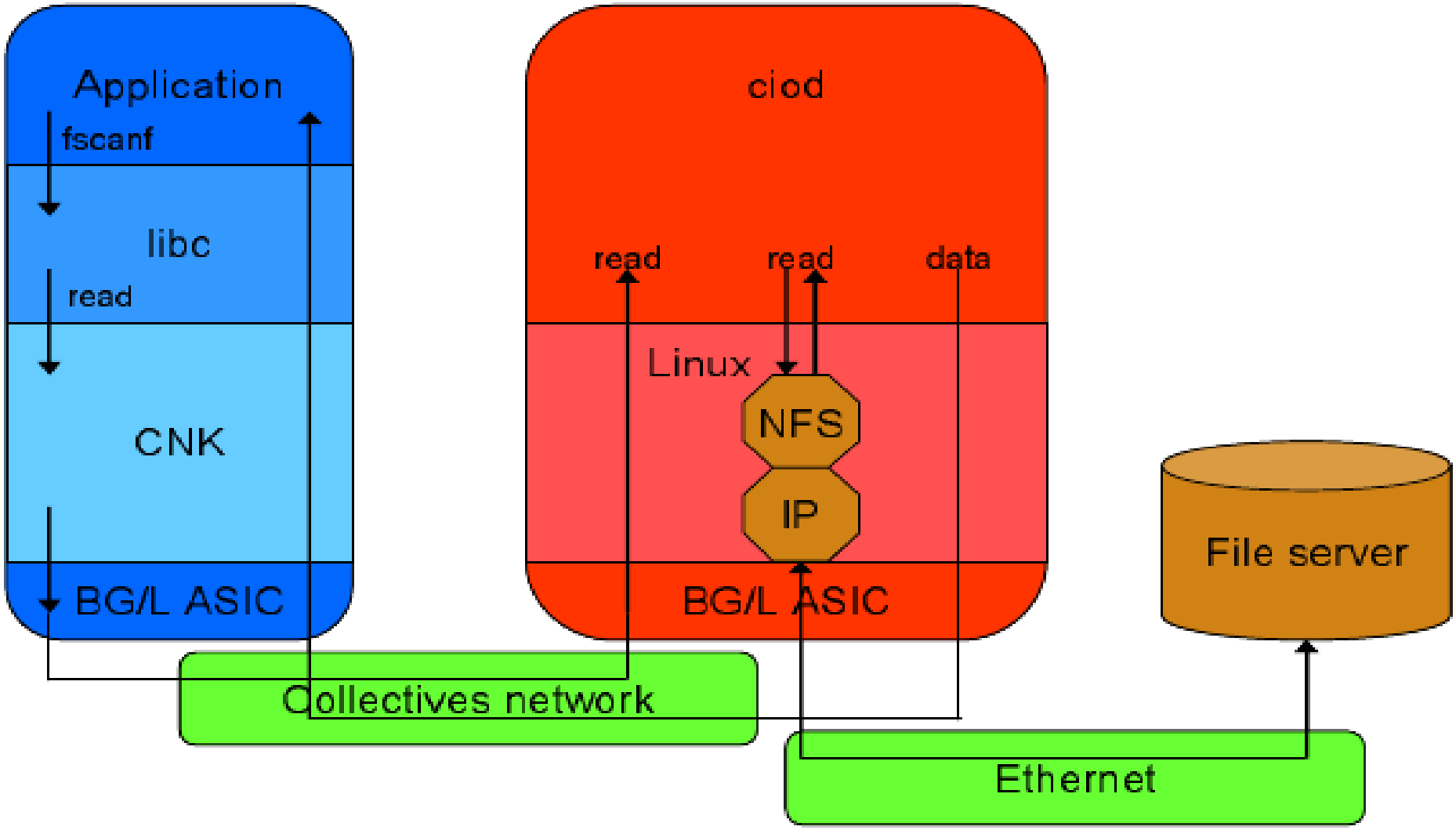


Figure 3: Function Shipping from CNK to CIOD.

Compute – I/O node interaction

- Separation of responsibilities allows for simplified compute node operation
- Also simplifies security issues
- Improves scalability, reproducability, and high-performance

Service Nodes

- In charge of control software
  - Operation and monitoring of all compute and I/O nodes
  - Also responsible for other hardware components
  - Setting up partitions, distributing code
  - Performs global network routing
  - Loads OS code

Node Personalities

Table 1: Personalities of Compute and I/O Nodes.

Compute node personality	I/O node personality
• Memory size • Bit-steering for memory configuration • Physical location • x, y, z torus coordinates • Pset number and size • Routes for the collectives network	• Memory size • Bit-steering for memory configuration • Physical location • Pset number and size • Routes for the collectives network • MAC and IP address • Broadcast and gateway IP addresses • Service node IP address • NFS server address and export directory • Security key

Experience / Experiments

- Socket Communication
- File systems
- MPMD applications
- Benchmarks / scalability

Socket Communication

- Traditional model uses MPI for all task communication
- Can it be adapted for use with sockets?
- Compute Node modifications

Socket Communication

- CIOD Modifications
  - Socket calls can potentially block indefinitely
  - Solution: use non-blocking calls
- Must strike a balance between polling existing processes and handling incoming requests

Socket Communication

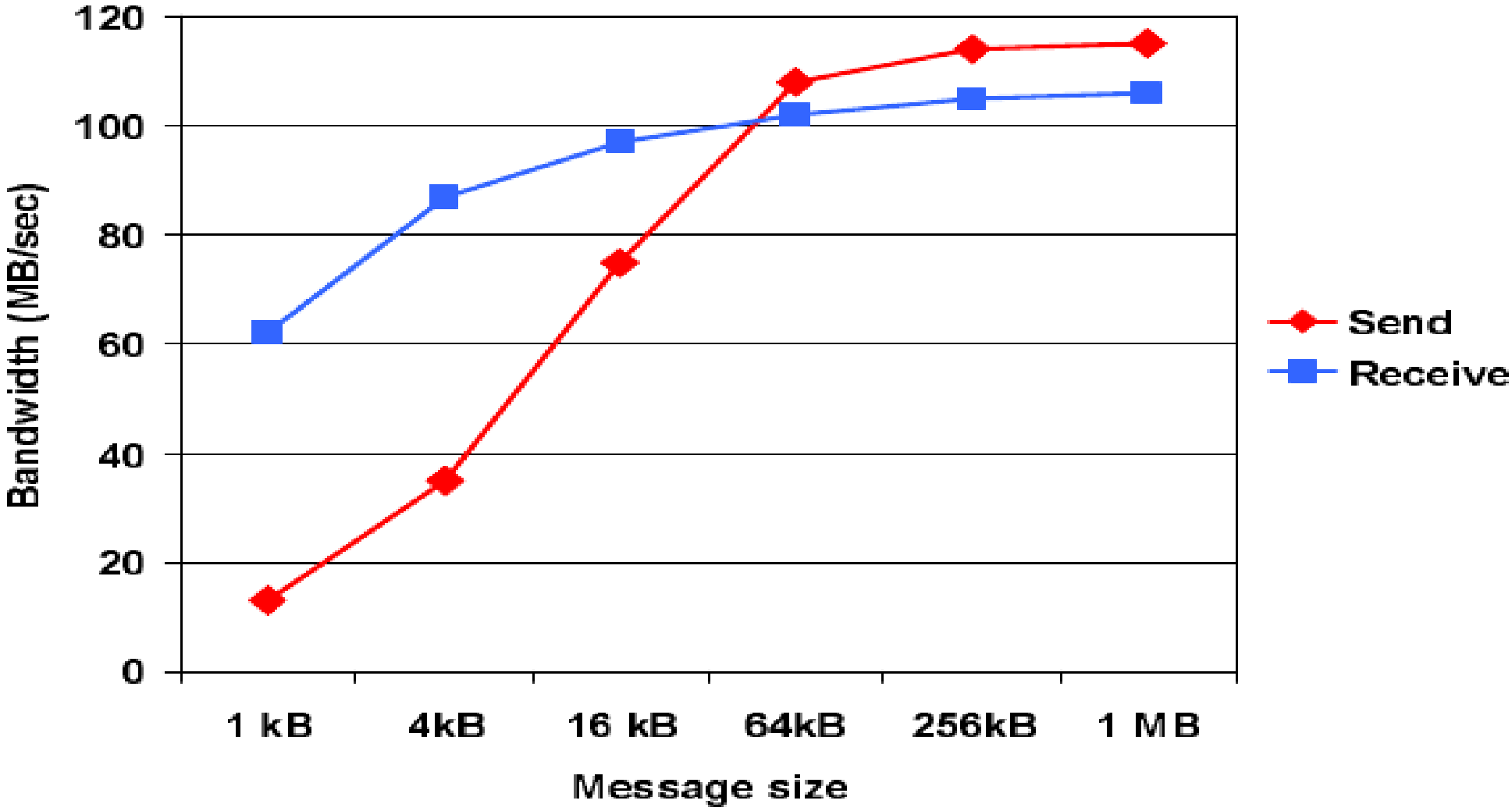


Figure 4: Performance of Socket I/O in Blue Gene/L.

File Systems

- Compute nodes are oblivious to the type of file system in use
- Porting file systems is straightforward
- Experiment
  - Blue Gene/L connected to GPFS cluster
    - 32 dual core 3.2GHz Xeon Servers
    - 8 disk controllers
    - 448 disks * 72GB / disc

File Systems

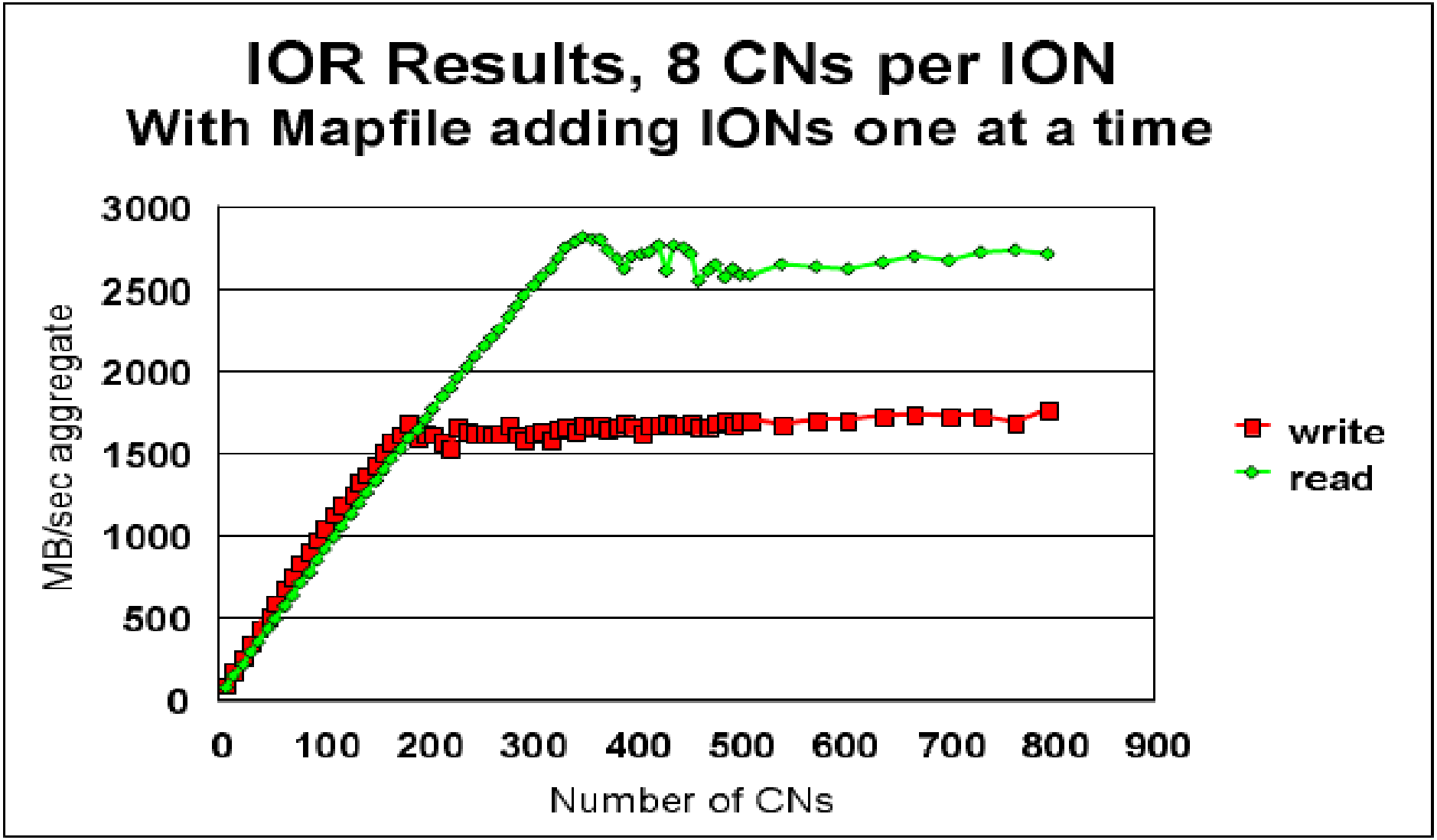


Figure 5: Performance of GPFS in Blue Gene/L.

MPMD Support

- Initially only supported SPMD applications

- Extended to include MPMD support

Benchmarks

- First experiment:
  - Single node, NAS benchmark
  - Two different operating systems

Benchmarks

Table 2: Single-node Performance for NAS Benchmarks on CNK and Linux.

	CNK		Linux			
	Time(s)	Speed (Mops)	Time(s)	Speed (Mops)	% Time	% Speed
is.A	5.7	14.7	23.0	3.6	303.0	-75.5
ep.A	172.6	3.1	177.2	3.0	2.7	-3.2
lu.A	380.3	313.7	504.0	237.0	32.5	-24.5
sp.A	375.0	227.0	517.4	164.3	38.0	-27.6
cg.A	27.1	55.0	32.3	46.3	19.2	-15.8
bt.A	522.6	322.0	613.0	274.4	17.3	-14.8

Benchmarks

- Second Experiment – Weak Scaling
  - Flash: astrophysics code

- Third Experiment – Strong Scaling
  - Miranda: hydrodynamics code

Benchmarks

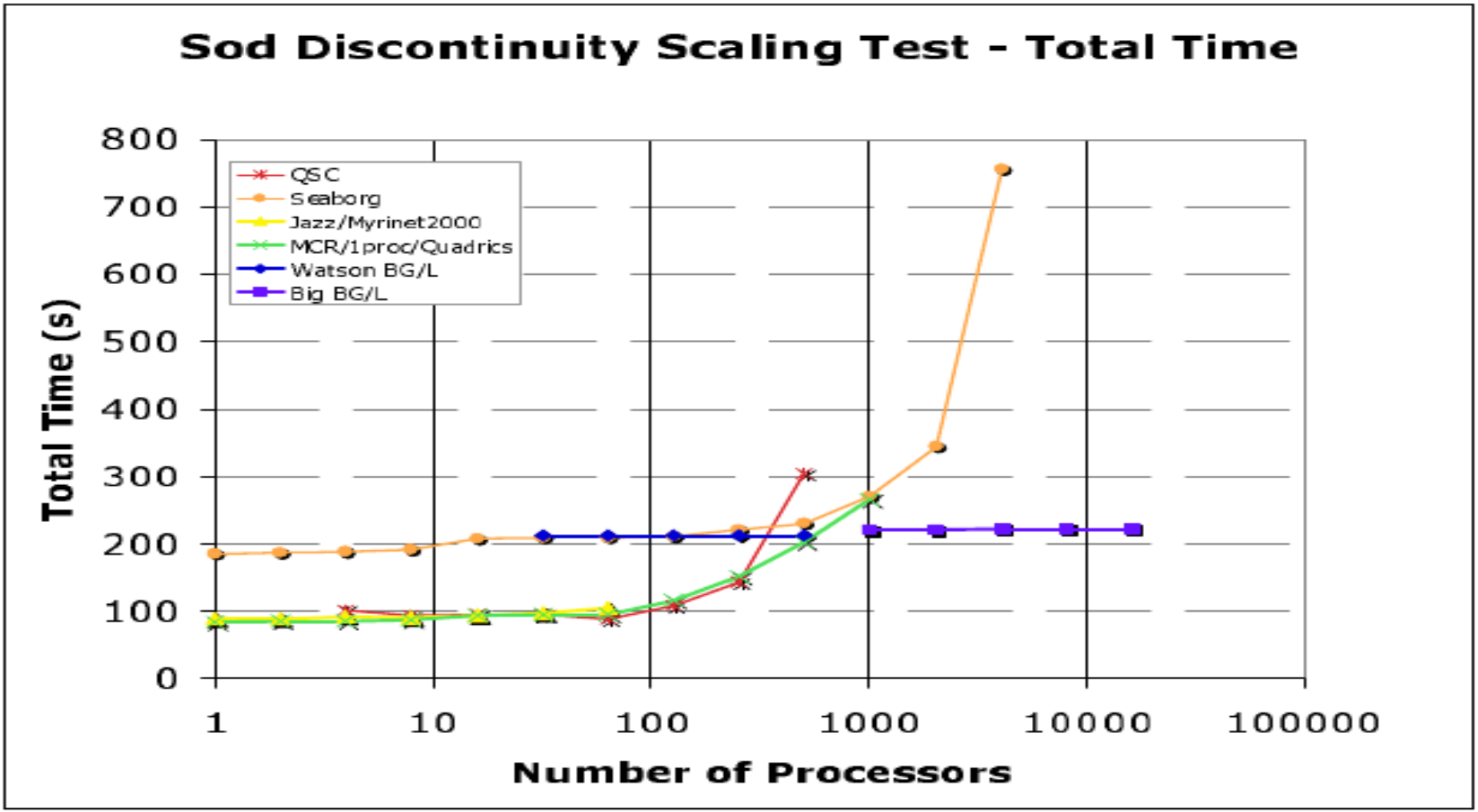


Figure 6: Performance of Flash in Different Parallel Systems (Weak Scaling).

Benchmarks

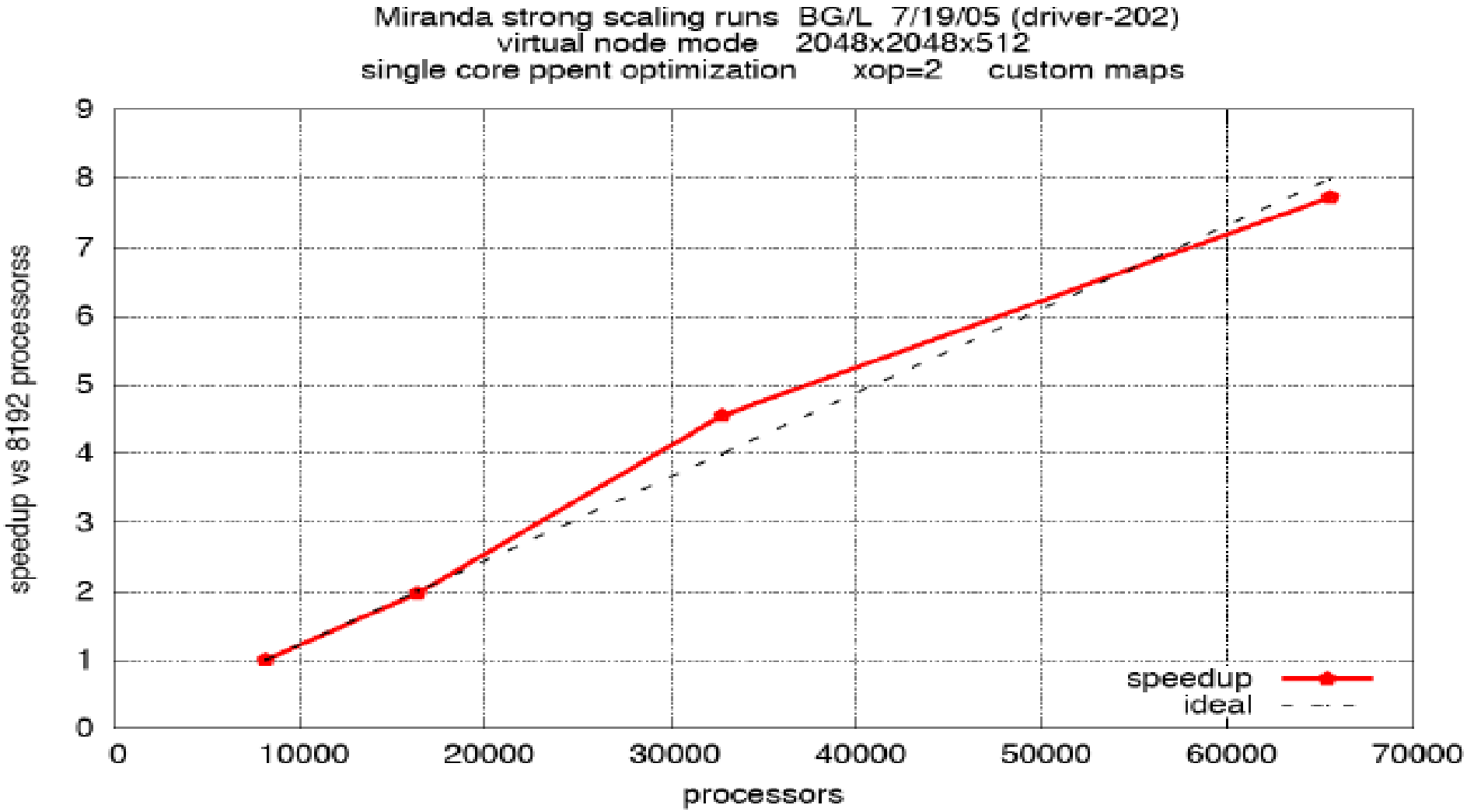


Figure 7: Strong Scaling of Miranda on Blue Gene/L (used with permission from the authors of [4]).

Conclusions

- OS design follows architecture design
- Separation of responsibilities has desirable qualities
- IBM continues to use this separation of roles in HPC designs

Barrelfish

- Premise: Multicore parallel processors have hit the mainstream, but OSes have not followed

- Solution: multikernel OS

Baumann, et. al., “The Multikernel: A new OS architecture for scalable multicore systems”

Barrelfish - Motivations

- Increasingly diverse systems
- Cores are increasingly diverse
- The interconnect matters
  - Inter-core communication is becoming a lot like message passing

Barrelfish - Motivations

- Messages cost less than shared memory

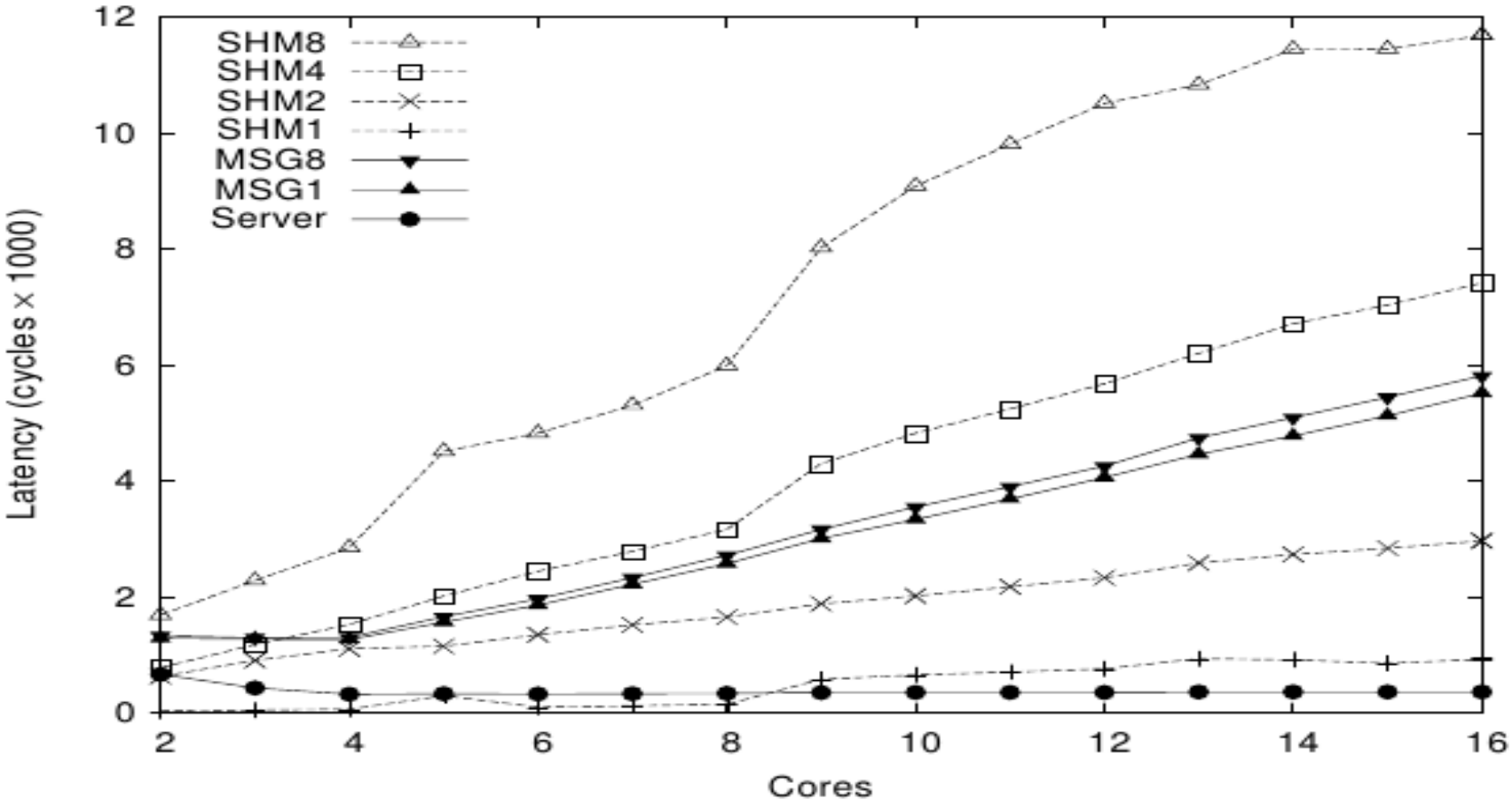


Figure 3: Comparison of the cost of updating shared state using shared memory and message passing.

Barrelfish - Motivations

- Cache coherence is not a panacea

- Messages are getting easier

The Multikernel Model

- Three design principles
  - Make all inter-core communication explicit
  - Make OS structure hardware-neutral
  - View state as replicated instead of shared

- Distributed system of cores that use messages to communicate
  - No shared memory

Explicit Communication

- Cache coherency and distributed shared memory systems obscure information
- Explicit communication behaves more like a network
- Non-blocking
- Makes analysis easier

Hardware-neutral OS

- Only two hardware specific considerations
- Easier to deploy across diverse systems
- Allows for optimization of message-based algorithms independent of hardware details

Replicated State

- OS state must be accessed by multiple processors
- Not necessarily saying the state MUST be replicated
- Can potentially even have a privately shared replica for different groups of cores

Applying the Model

- Some sacrifices need to be made
  - What to do with a shared L2 cache?
  - Replica consistency isn't free
  - Latency of messages becomes an issue
  - Increased burden on developers to understand and control consistency

Applying the Model

- Goals:
  - Comperable performance to modern OSes
  - Demonstrates scalability
  - Can be re-targeted to different hardware without refactoring
  - Exploit MP optimization techniques
  - Exploit OS modularity to place functionality based on hardware topology or load

Implementation – Test platforms

- 2X4 core Intel system
  - 2.66GHz, shared 4MB L2 cache
- 2X2 core AMD system
  - 2.8GHz, 1MB L2 cache per core
- 4X4 core AMD system
  - 2.5Ghz, 512kB L2 / core, 6MB shared L3 / 4 cores
- 8X4 core AMD system
  - 2GHz, 512kB L2 / core, 2MB shared L3 / 4 cores

Structure

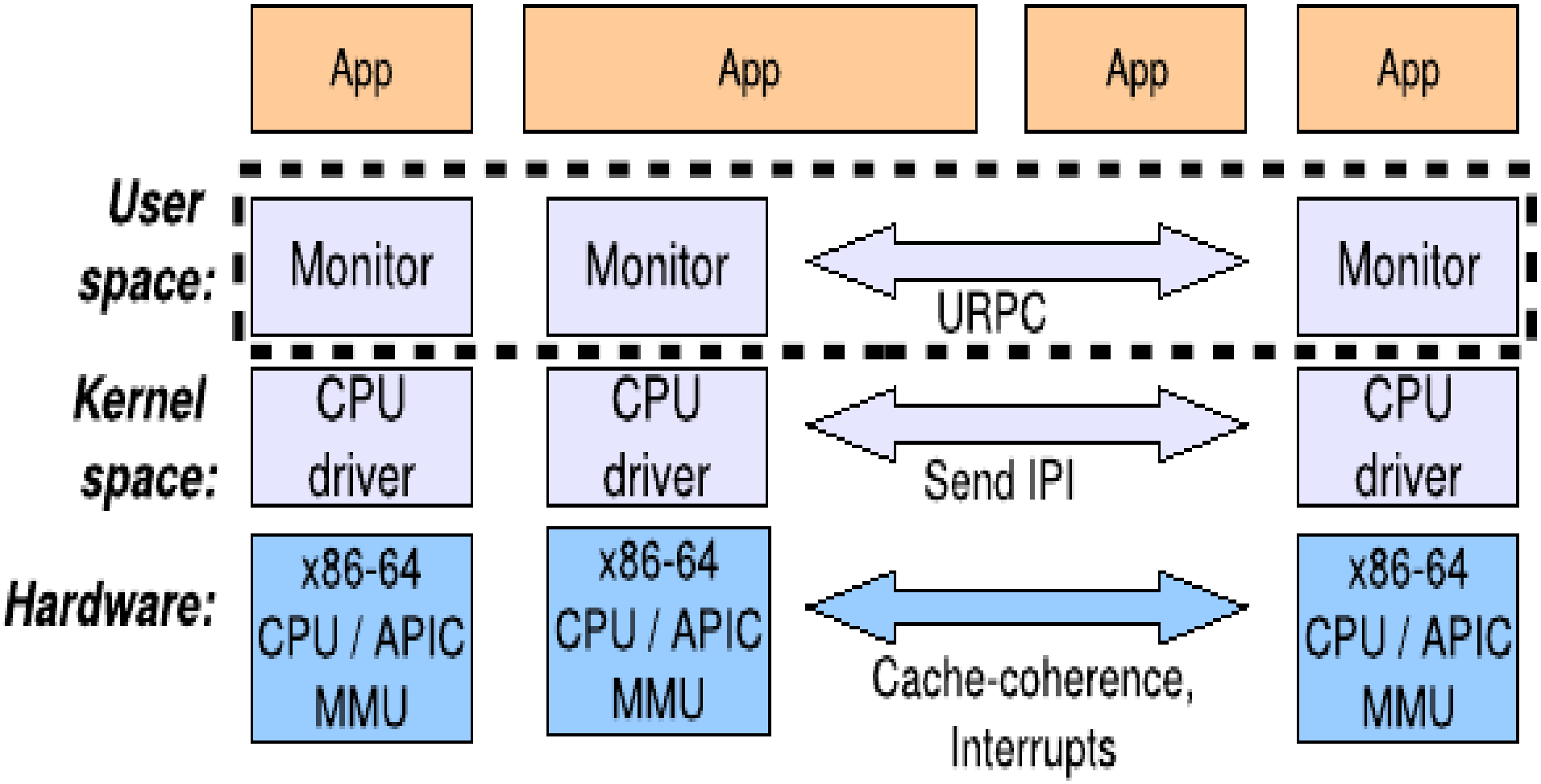


Figure 5: Barrelfish structure

CPU Drivers

- Roles: Authorization, time slicing, hardware access
- Shares no state with other cores
  - Event driven, single-threaded, nonpreemptable
- Small: 7k lines of C, 300 lines of assembly
- Has the most direct access to hardware
  - Most hardware specific piece of the OS

Monitors

- Encapsulate much of the mechanisms and polices of a traditional kernel
- Single core, user-space, schedulable
- Replicated data structures kept consistent by an agreement protocol
- Responsible for setting up interprocess comm.

Process Structure

- Processes are represented by a collection of dispatchers

- Dispatchers are schedule by the CPU driver

Inter-core communication

- All inter-core communication occurs with messages
  - Different mechanisms for different scenarios

- Cache-coherent memory:
  - Based on user-level RPCs (URPCs)
  - Use shared memory as a channel
  - Cache line sized messages
  - Point-to-point

Inter-core communication

- Implementation:
  - Sender writes message sequentially to cache
  - Receiver polls last word of the line (why?)
  - Abstracted behind an interface

- Polling forever is inefficient

Memory Management

- Physical Memory is still a global resource
  - OS must manage it even if it doesn't use it directly
- Barrelfish uses a capability system
  - Memory is managed at the user level by altering capabilities of memory regions
  - Example: RAM capabilities, page table capabilities, etc.
  - CPU driver's role is to ensure correctness
    - Memory location exists, no overlap, etc.

Shared address spaces

- There is still a great need to support shared memory
- Barrelfish supports this by coordinating dispatchers
  - Virtual address space
  - Capabilities
  - Thread management
- Message passing is used to keep data consistent among the dispatchers sharing resources

Knowledge and Policy Engine

- System knowledge base (SKB)
  - Manages hardware knowledge
- Uses this data to optimize the system
  - Selecting message transports
  - Topology aware resource management

Evaluation

- TLB shootdown
- Messaging performance
- Compute-bound workloads
- IO workloads

TLB Shootdown

- Chosen because it is a simple operation but still requires global coordination
  - Must maintain TLB consistency by invalidating entries when pages are unmapped
- Linux and Windows use interrupts

TLB Shootdown

- Barrelfish uses messages
- In general this has higher latency than interrupts
- Four approaches tried

TLB Shootdown

- Broadcast
- Unicast
- Multicast
  - Message sent to first core, which passes the message on
- NUMA-Aware
  - Multicast with a twist

TLB Shootdown

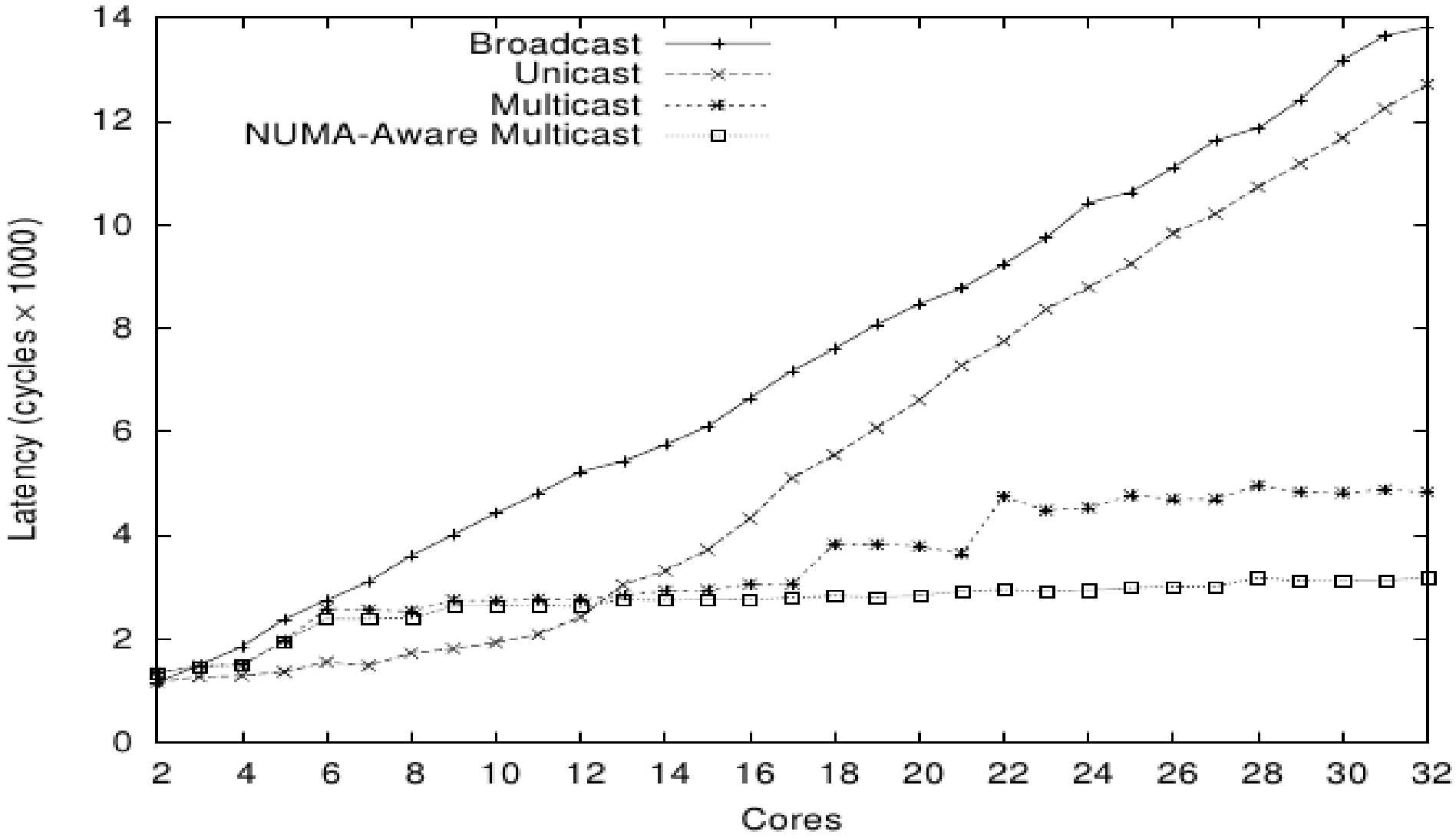


Figure 6: Comparison of TLB shootdown protocols

TLB Shootdown

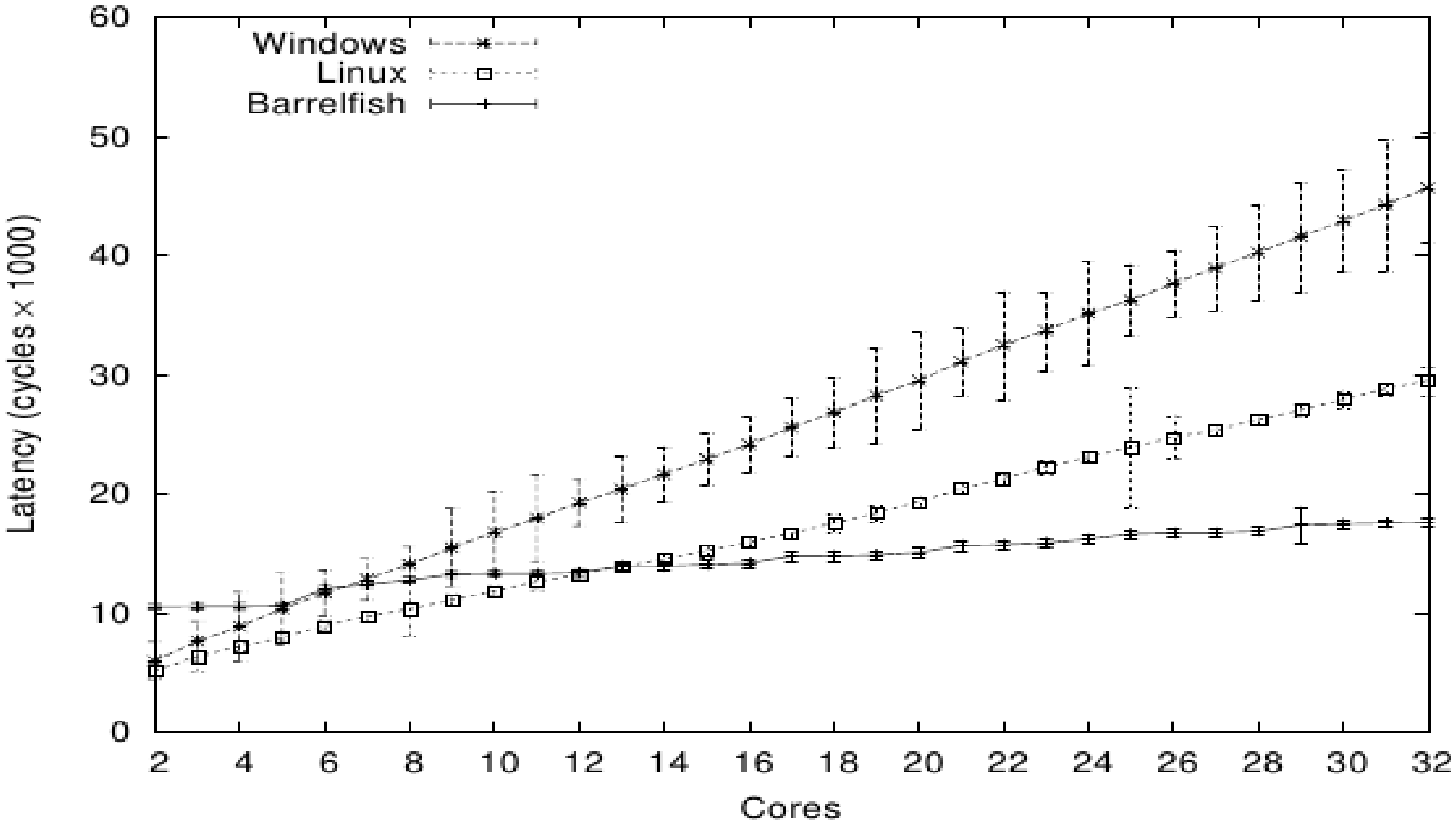


Figure 7: Unmap latency on 8×4-core AMD

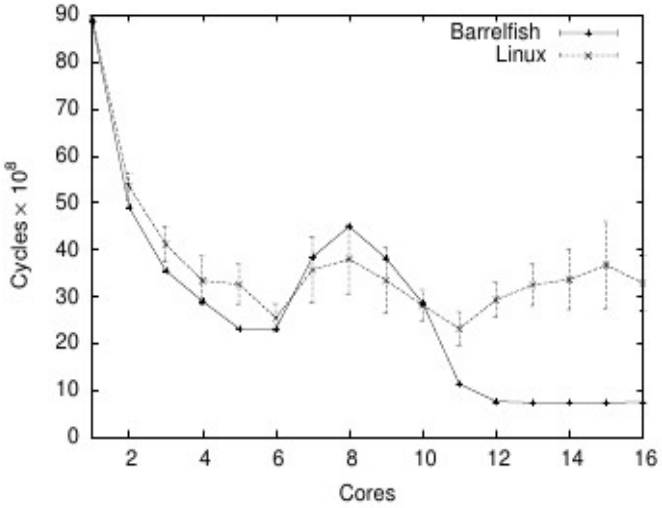
Messaging Performance

- Polling: How wasteful is it?

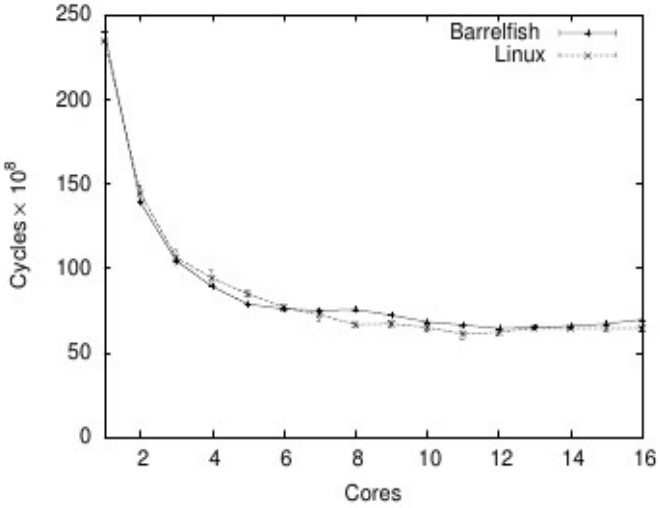
$$overhead = \begin{cases} t & \text{if } t \leq P, \\ P + C & \text{otherwise.} \end{cases}$$

$$latency = \begin{cases} 0 & \text{if } t \leq P, \\ C & \text{otherwise.} \end{cases}$$

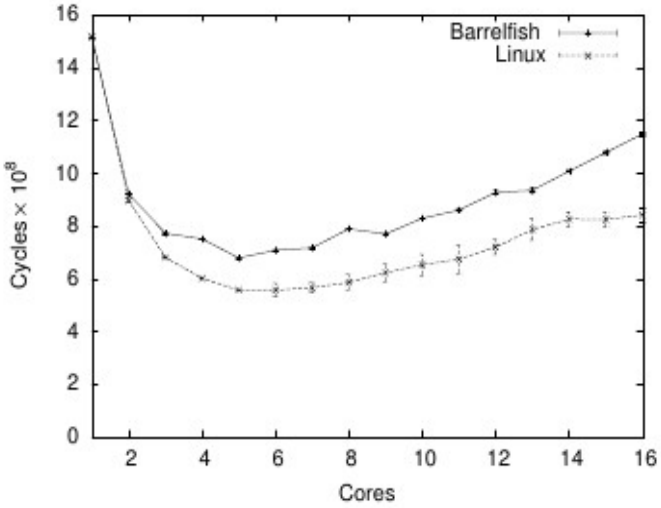
Compute-bound Workloads



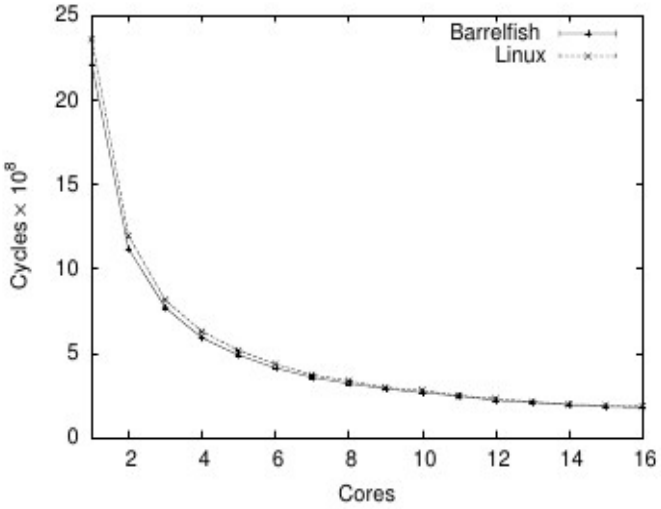
(a) OpenMP conjugate gradient (CG)



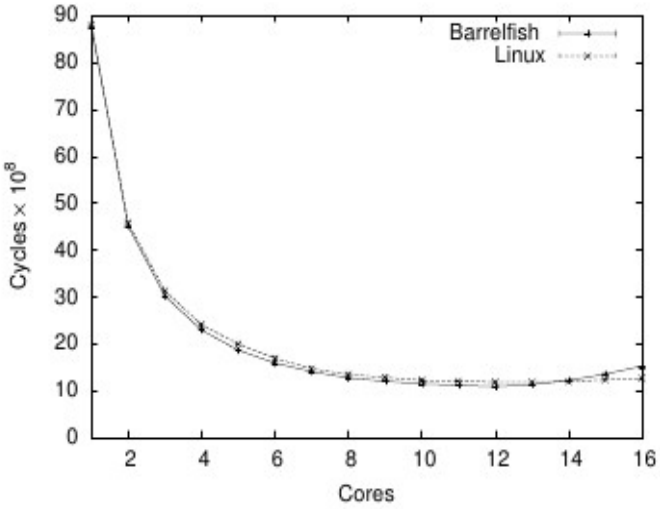
(b) OpenMP 3D fast Fourier transform (FT)



(c) OpenMP integer sort (IS)



(d) SPLASH-2 Barnes-Hut



(e) SPLASH-2 radiosity

I/O Workloads

- Web server
  - Server software, NIC driver, system services on separate cores
    - Barrelfish: 18697 requests / sec
    - Linux: 8924 requests / sec

Summary

- Results not intended to say that Barrelfish performs better or worse than existing OS
- Still an infant OS – much more to come

Implementations

- X86
- ARM
- Intel SCC / Knight's Ferry
- Beehive

HPC applications?

- Stated design: general purpose
- But think about the other HPC OS we looked at
  - Or just take a look at the TOP500 list
- Could HPC be the ultimate goal?