Corey: An Operating System for Many Cores

S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y. Dai, Y. Zhang, Z. Zhang

OSDI 2008

Presented by Zachary Bischof

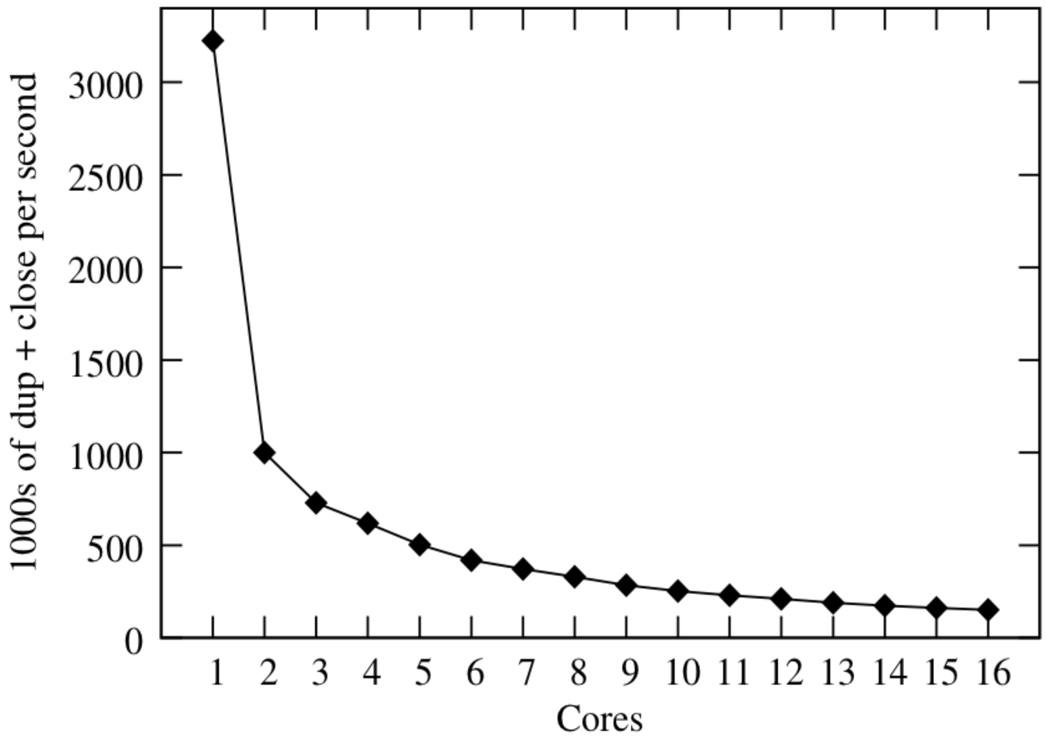
NORTHWESTERN
UNIVERSITY

Background

- Focus of chip manufactures has been number of cores on a chip
  - This leads to more sharing between cores
- Modern operating systems not optimized for sharing between cores
  - Sharing between cores may not be required
  - Unnecessary sharing becomes a bottleneck for performance
  - Example: File descriptors

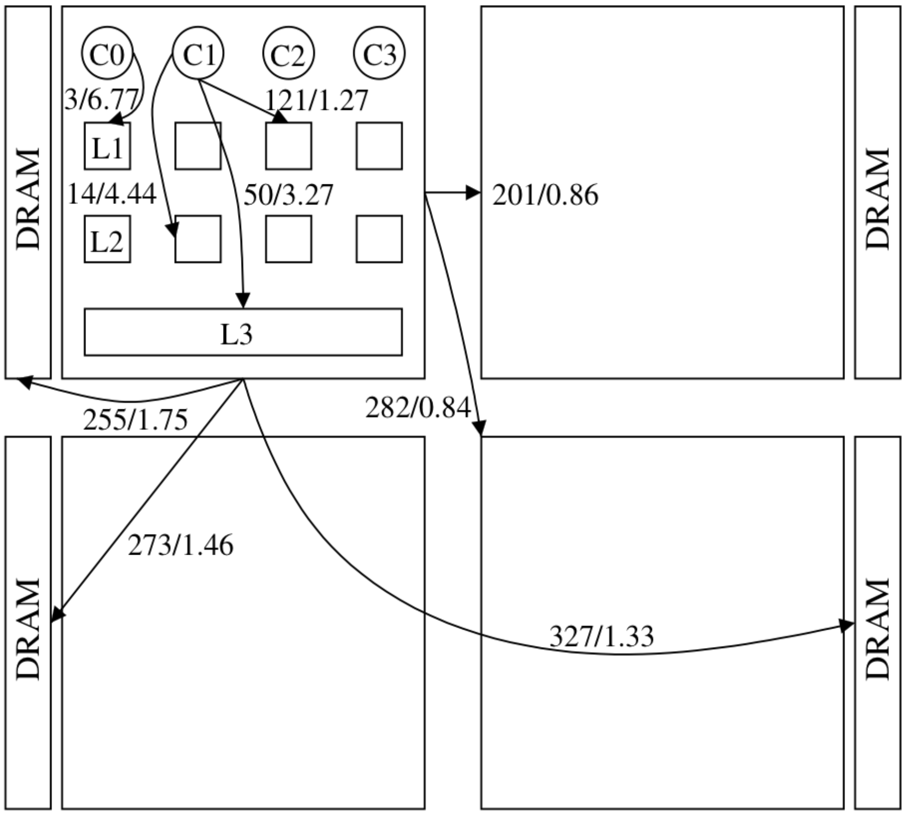
dup and close

- As number of cores increases, dup + close operations decrease
- Shared table describing open files is causing the contention
- Standard is that all new file descriptors must be visible to all threads



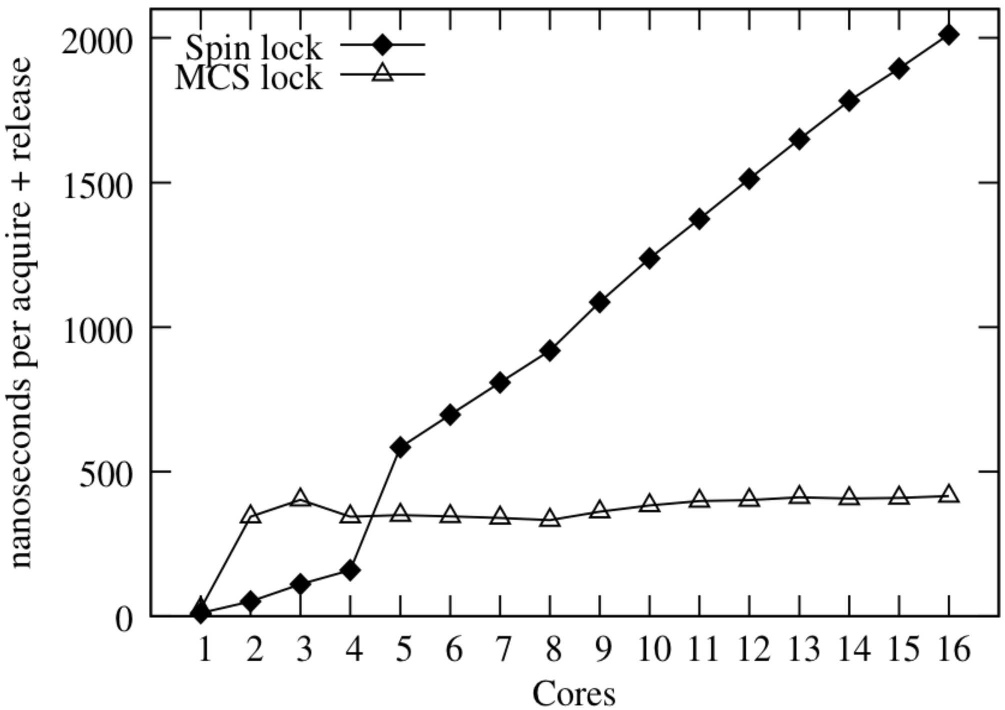
Explanation

- How falsely shared cache line hurts performance
  - Inter-chip reads are slow
  - Sharing requires accesses to remote cache



Explanation

- Widely-shared locks decrease performance
- Corey uses MCS locks where each core spin separately



NORTHWESTERN UNIVERSITY

Proposed Solution: Corey

- Want to allow applications to scale well with an increase in the number of cores
- Try something new (like an exokernel…)
  - Provide abstractions that applications can control
  - Applications can control sharing of OS data structures
- Corey's new abstractions for the OS
  - Address ranges
  - Kernel cores
  - Shares

Address Ranges

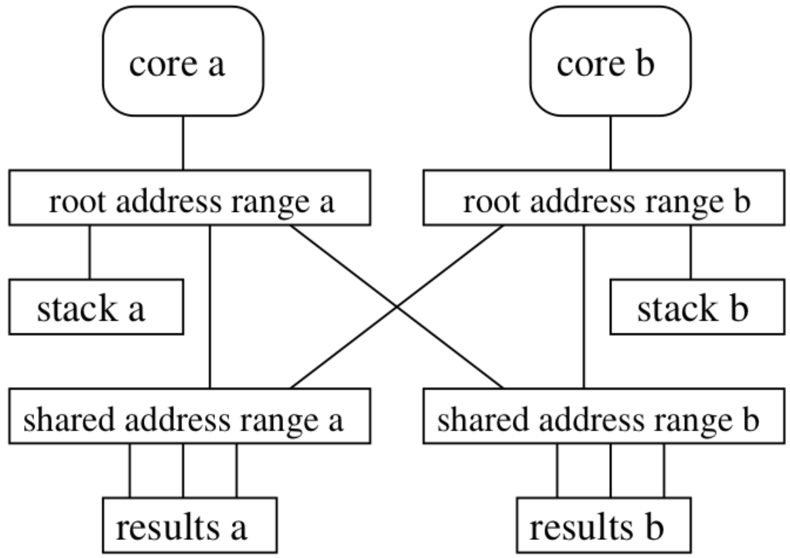
- Options for concurrency
  - Threads
    - Typically shares a single address space between all threads
  - Processes
    - Typically has separate address spaces for each process
  - Each only works for one sharing pattern
- Applications wanting a mix of both are forced to choose (e.g. MapReduce)

MapReduce

- Two phases
  - Map phase
    - Master node takes input, splits up the work, distributes to other nodes (this process is repeated by worker nodes)
    - Separate address spaces has no contention
    - Shared address causes contention when distributing data
  - Reduce phase
    - Master node takes the answers to sub-problems and combines them to get output (repeated up the chain)
    - Separate address space leads to soft page faults per core per page of intermediate results
    - Shared address space has no soft page faults as results are returned
- We want the best of both worlds

Address Ranges

- Corey's kernel abstraction of address ranges
  - Range of virtual-to-physical mappings
  - An application can allocate ranges, insert mappings, and place an address range at the desired location
  - If multiple cores' address space uses the same range, the space is shared
- Result
  - A core can update private address space without contention
  - Space is only shared with cores that manipulate the mappings



(c) Two address spaces with shared result mappings.

NORTHWESTERN UNIVERSITY

Kernel Cores

- System calls in applications
  - System calls are performed on same core as caller
  - Must acquire locks for shared kernel data structures
  - Can be costly
- Kernel abstraction for a kernel core
  - A single core handles all kernel functions
    - Manages hardware devices
    - Execute system calls from other cores
  - E.g. A Web service application with a core dedicated to handling the network device
- Application decides if there will be performance improvements

Identifier Sharing

- Many kernel operations need to look up identifiers in tables to find a pointer to kernel data
  - File descriptors
  - Process IDs
- The OS implementation determines the scope of sharing of identifiers and tables (e.g. Unix)
  - File descriptors shared between threads
  - Process identifiers are generally global

NORTHWESTERN UNIVERSITY

Shares

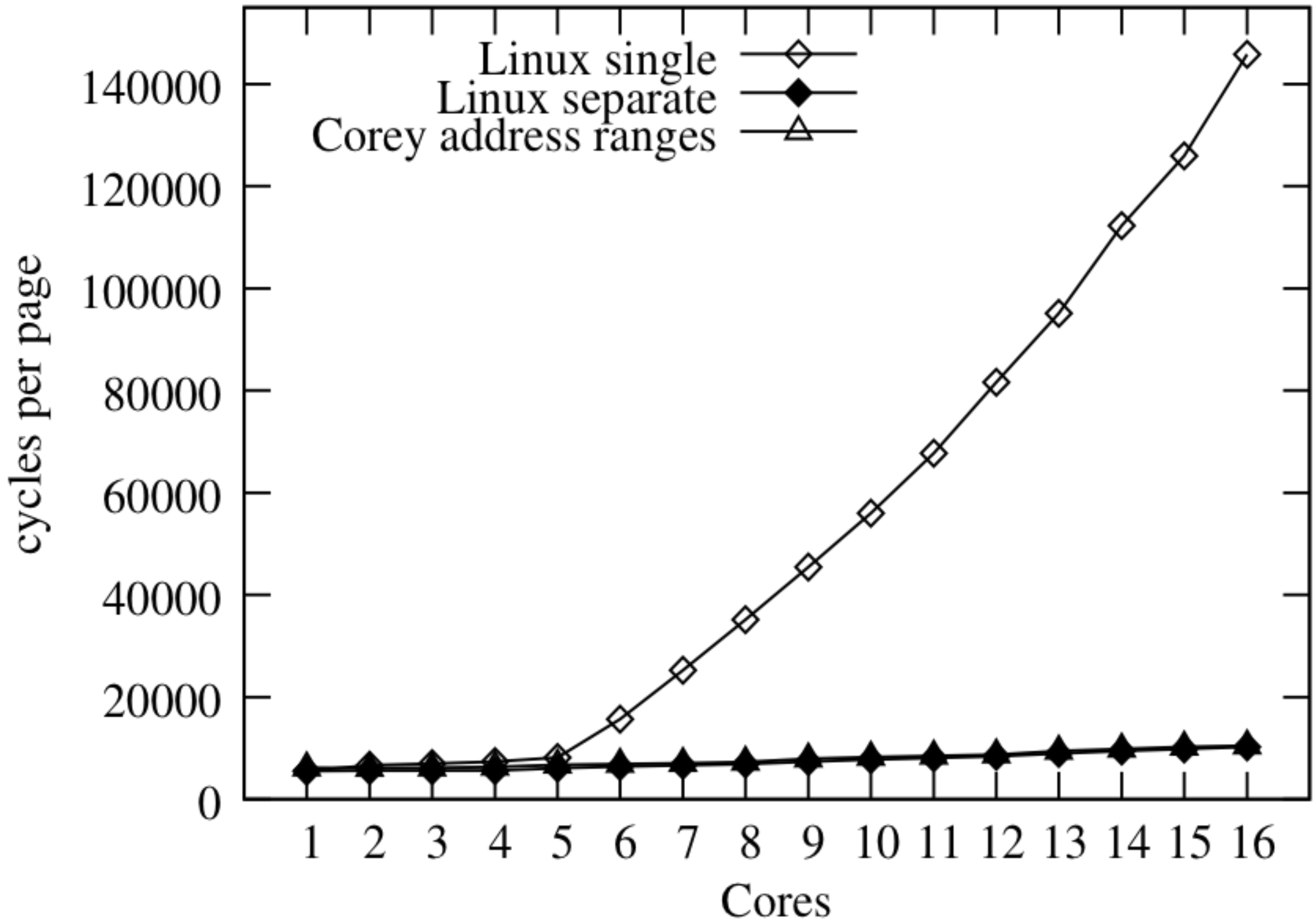
- Kernel share abstraction
  - Allow applications to create lookup tables and control sharing
    - Each core starts with a unique, private share
    - Sharing is done by creating a share and adding the share's ID to that core's private root share (or a share within the root share)
    - A root share is always private and does not need locking
    - The shares that are reachable from the private share are the identifies the core can use
  - Contention may still be a problem but is avoidable
    - Identifiers should always be placed in most limited sharing
  - Applications must keep track of the location of identifiers

NORTHWESTERN UNIVERSITY

Back to File Descriptors

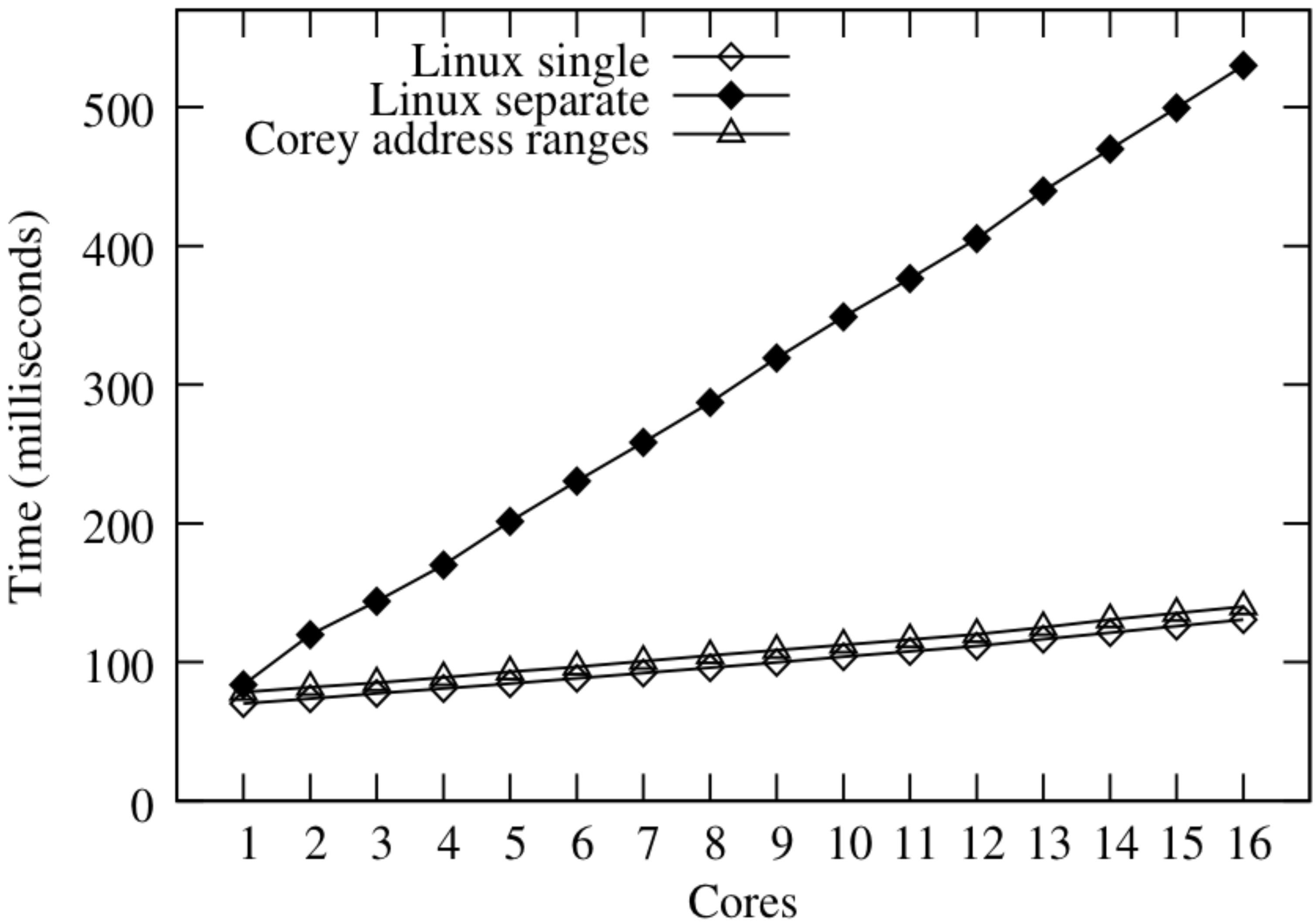
- Private file descriptors
  – Place descriptors in its core private root share if it is only used by one thread
- Shared file descriptors
  – All cores sharing the descriptor create a share that holds the descriptor
- Application can limit sharing and avoid unnecessary contention between tables and identifiers

Performance (Address Ranges)



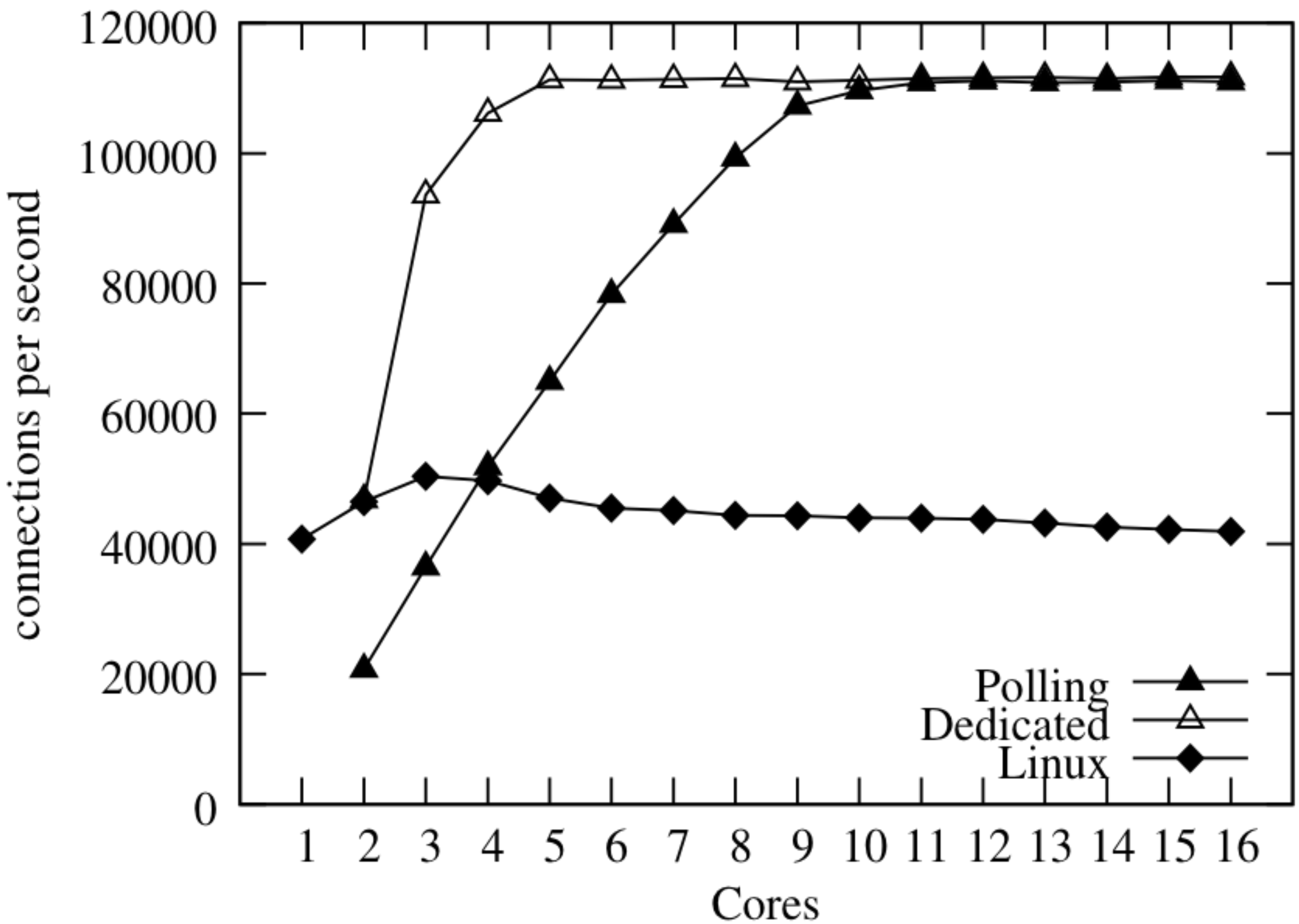
(a) memclone

Performance (Address Ranges)



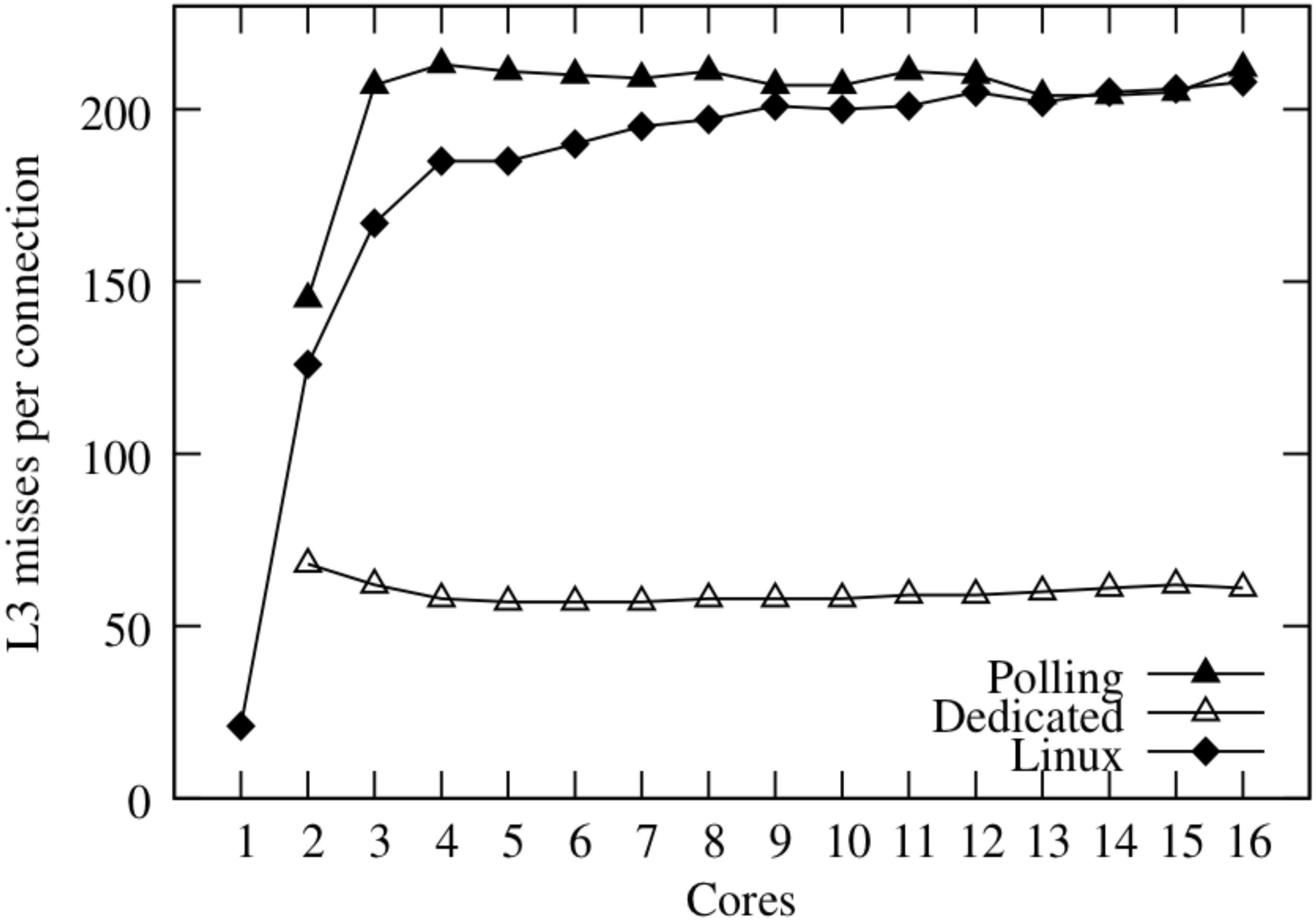
(b) mempass

Performance (Kernel Cores)



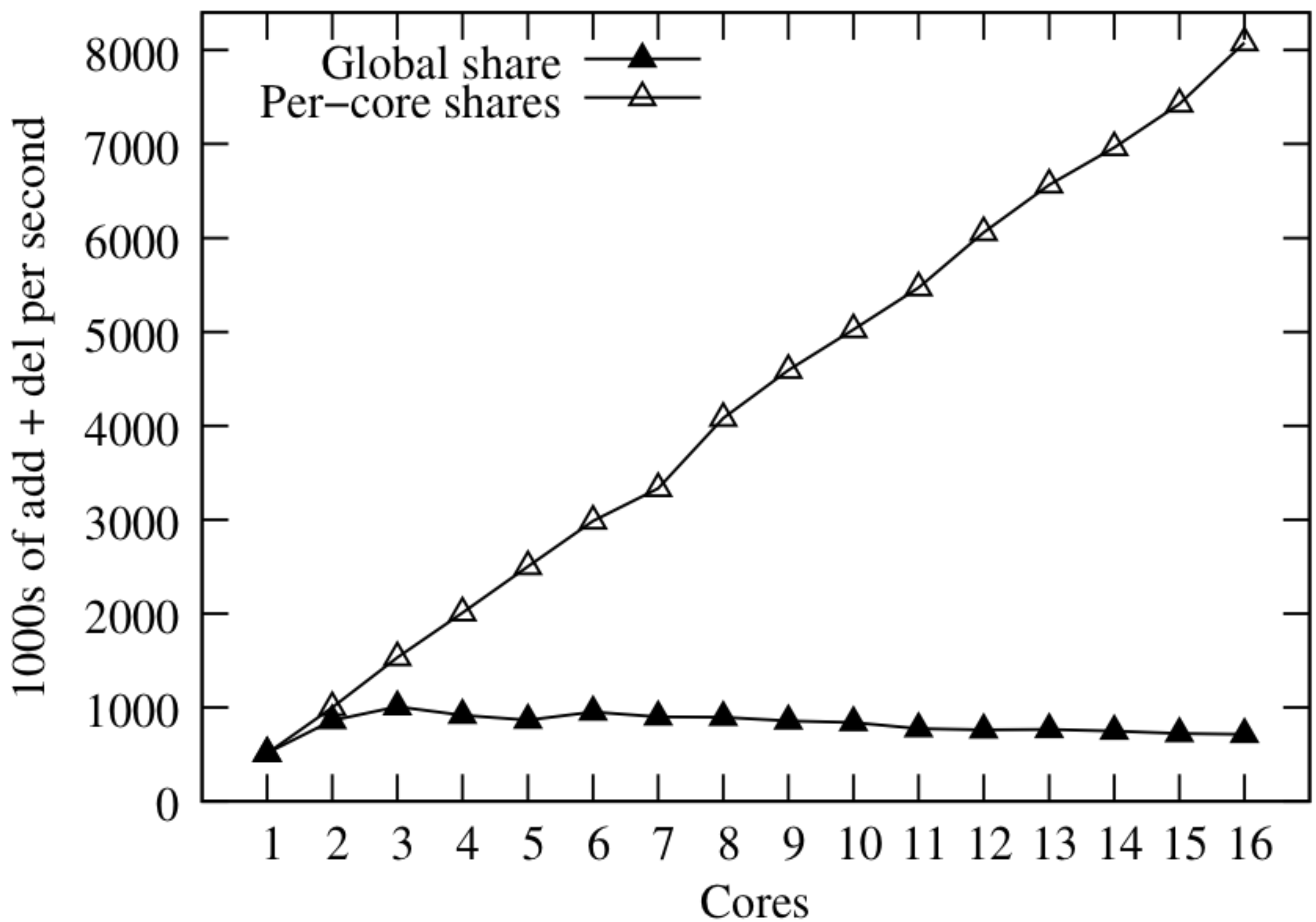
(a) Throughput.

Performance (Kernel Cores)



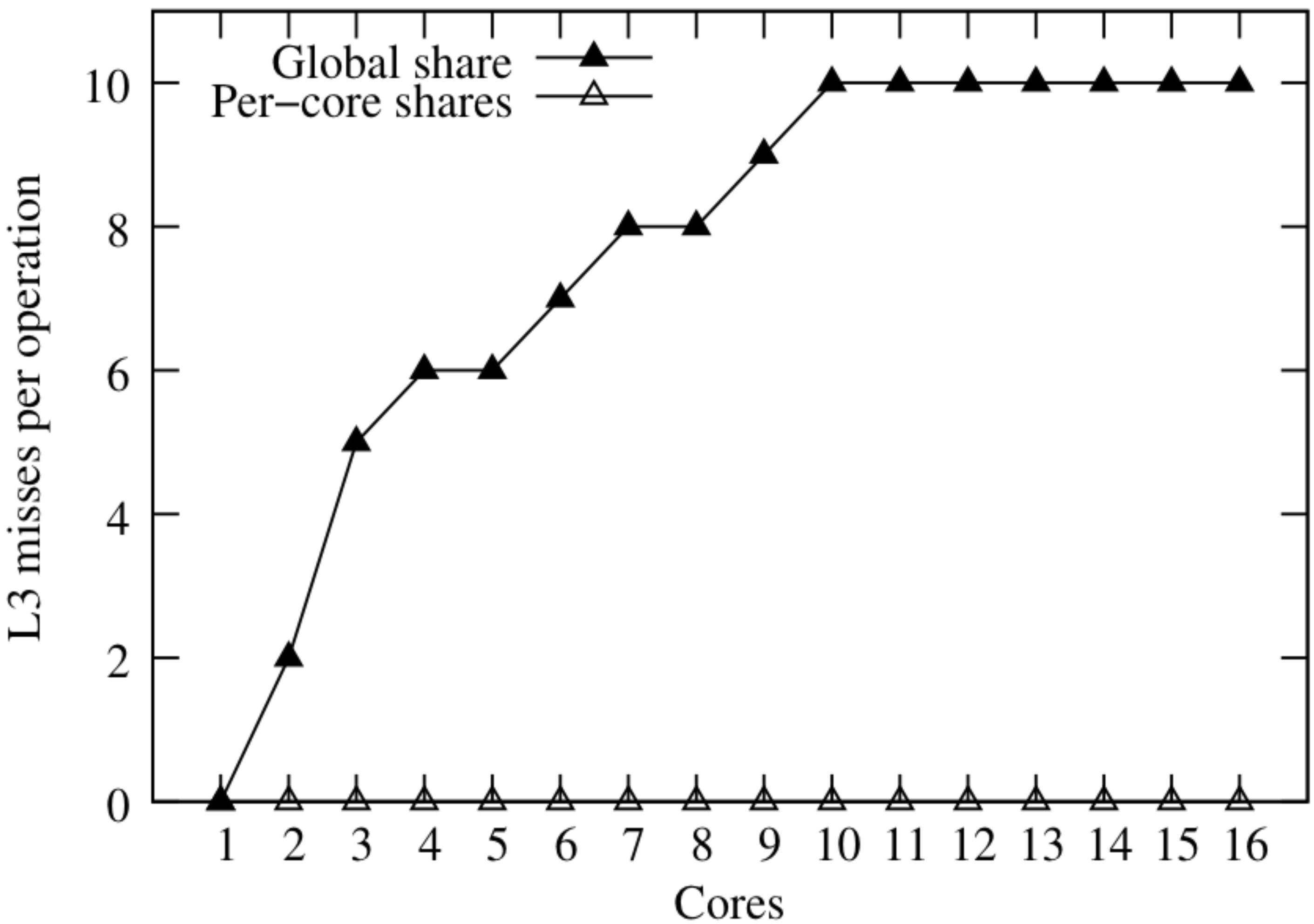
(b) L3 cache misses.

Performance (Shares)



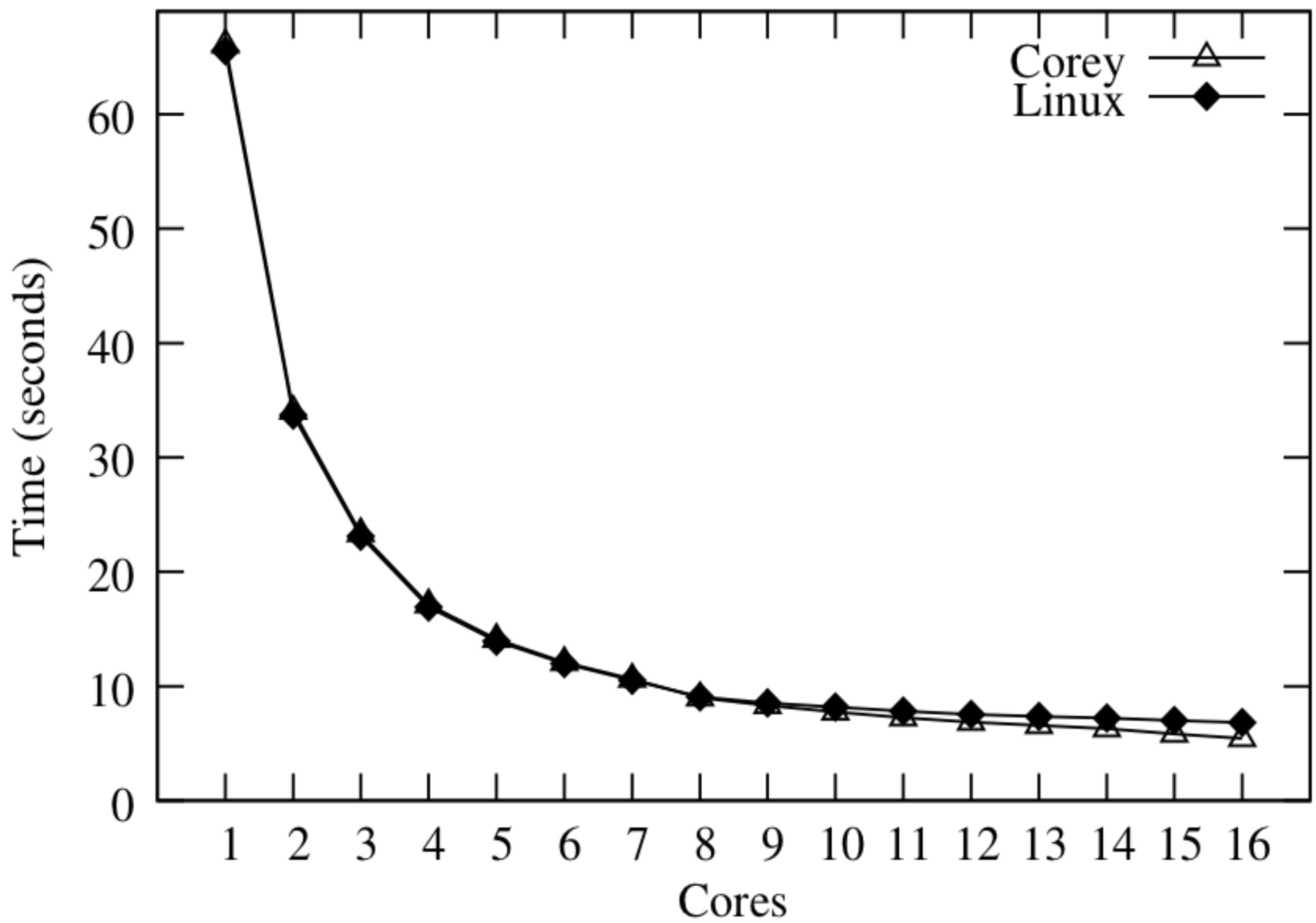
(a) Throughput.

Performance (Shares)



(b) L3 cache misses.

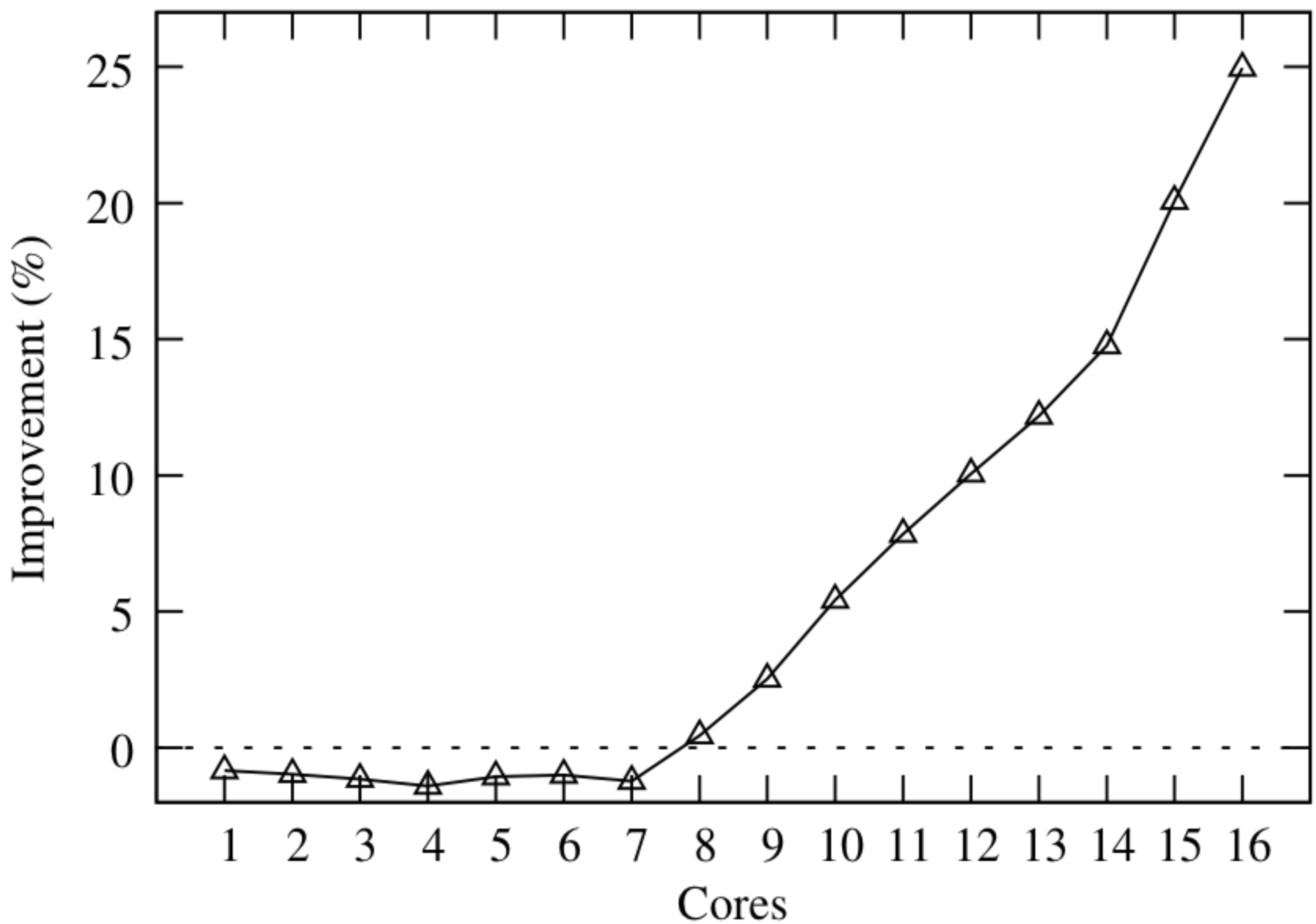
Performance (wri MapReduce)



(a) Corey and Linux performance.

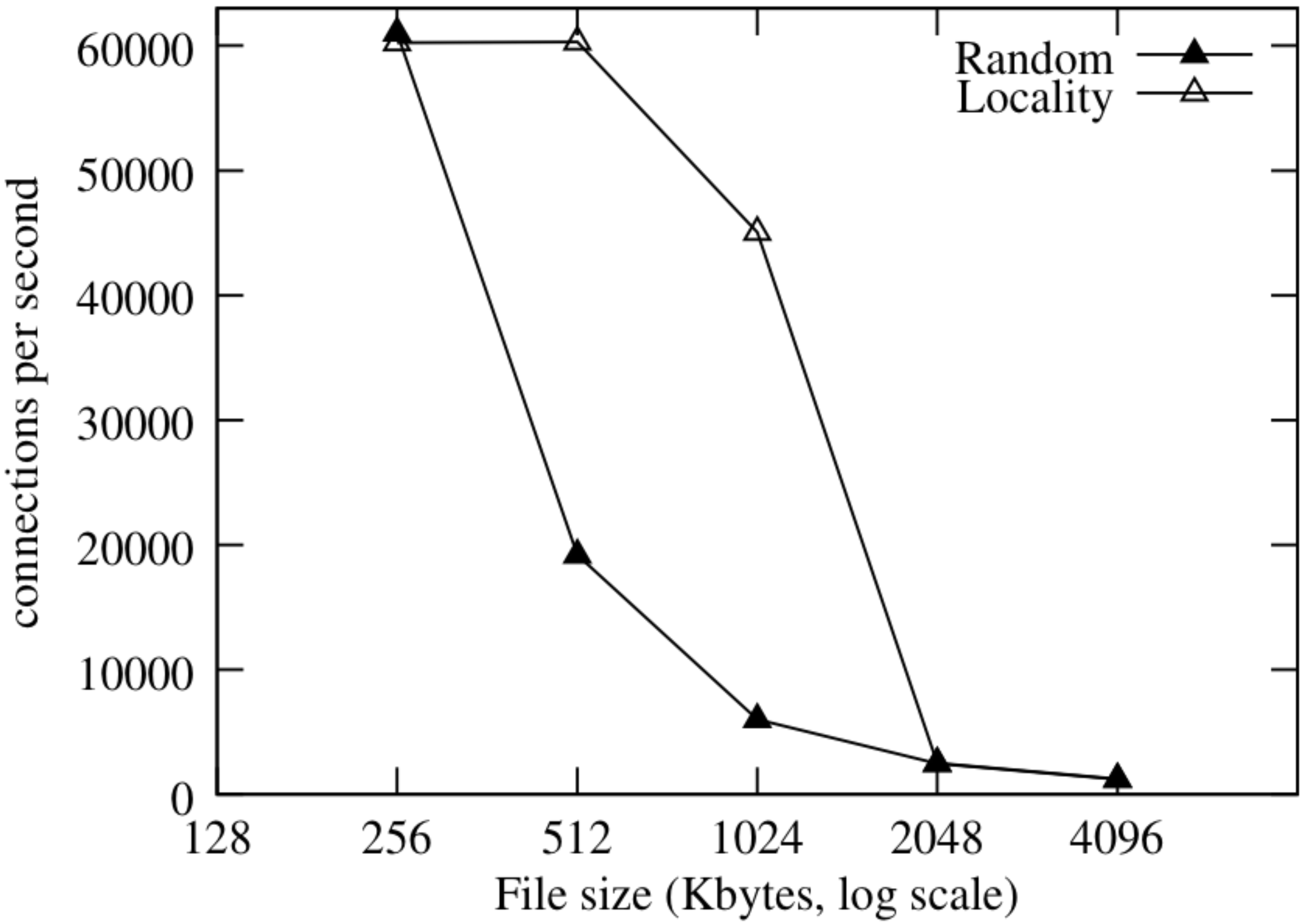
Performance (wri MapReduce)

(a) Corey and Linux performance.



NORTHWESTERN UNIVERSITY

Performance (webd)



NORTHWESTERN UNIVERSITY

Comments

- Corey is a prototype
  - May not be a fair comparison to Linux
  - Actual performance could be affected both ways
- Many of these concepts could be implemented current Oses
- Paper is trying to argue that applications need to control sharing for scaling purposes
  - Exokernels may become more important as the number of cores per chip continues to increase