

# Will the I/O model survive till the 22nd century?

Rasmus Pagh  
IT University of Copenhagen

# This talk

- This is a **workshop talk**, not a report on research results.

## Outline:

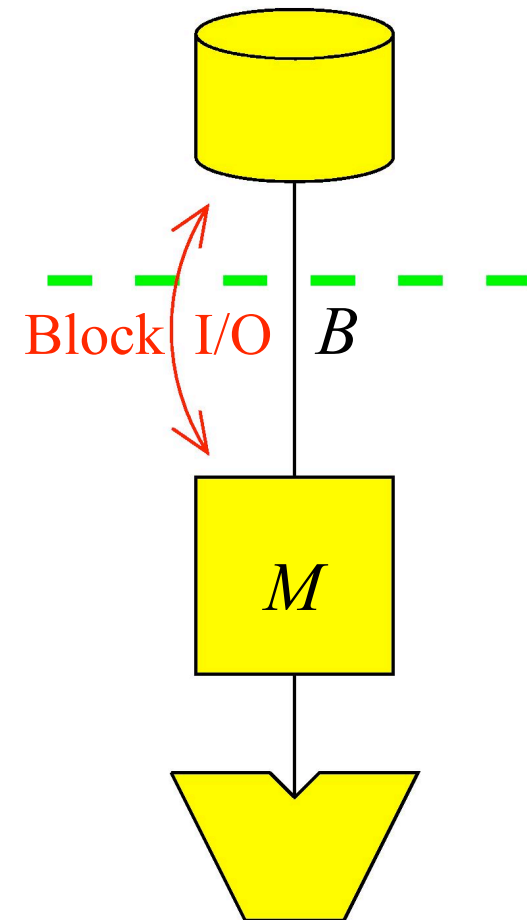
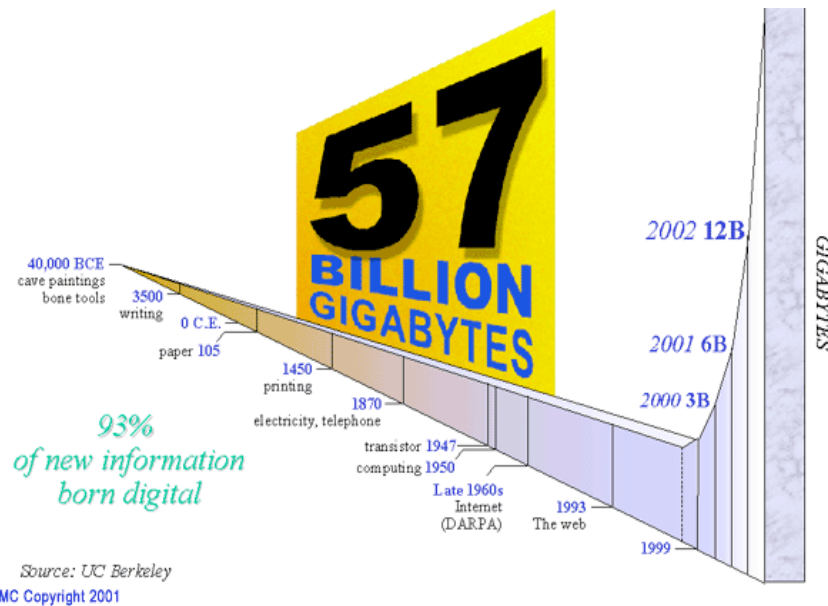
- Features of the I/O model.
- A look at justification of the I/O model (for 22nd century amounts of data).
- sI/O model.
- Case studies: Sorting and BFS.
- Conclusion and open questions.

# Disclaimer

- I do not claim to be able to answer the question asked by the title. In fact I'm neither:
  - A hardware designer, or
  - A physicist, or even
  - An active researcher in I/O algorithms.
- Many points in the talk may be subject to discussion — provoking such discussion is indeed the hope I have in giving this talk.

# The I/O model

- Beautifully simple.
- Excellent in terms of reflecting real-world performance of algorithms working on large data sets.



# Cache-obliviousness

- Can be viewed as a property of algorithms, namely independence of  $M$  and  $B$ .
- Analysis still uses I/O model (along with some assumption on the caching used).
- The rest of the talk will deal with the classic I/O model.

# Why block transfers?

## - the hardware explanation

- Disks have latency: When a piece of information is requested, it takes time  $t$  to get the first bit.
- Using additional time  $t$  to transfer a sequence of adjacent bits on the disk only loses a factor of 2, and in many cases gives a speedup proportional to  $t$ .

# Why block transfers?

## - the physics explanation

- Need to access data that is physically located away from the processor.
- Time  $t$  to retrieve a single bit grows (at least) linearly with distance, as the speed of transmission is bounded by  $c$ .
- As before, transmitting further bits for time  $t$  at most doubles access time.

# Why rounds of communication?

Why do we wait to receive a block before requesting the next?

- It would at most give a constant factor improvement, as we already spend a constant fraction of the time transmitting data (ignoring computation time).

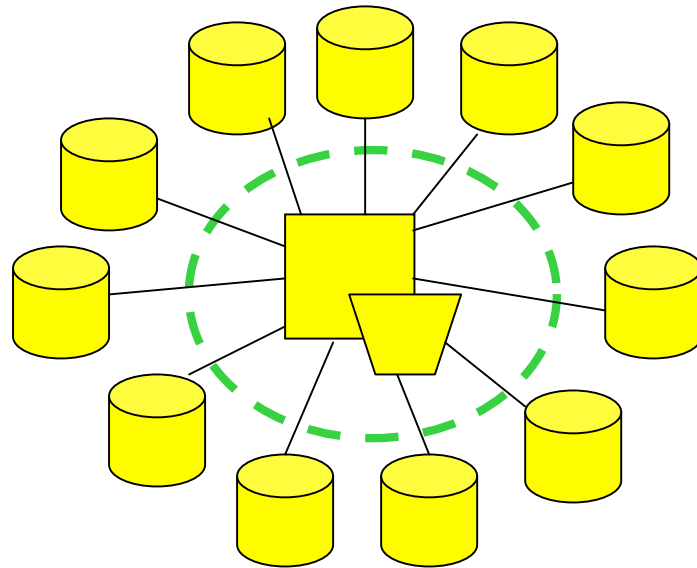
# Why spatial locality?

Why are the blocks we read consecutive entries in the memory array?

- **Hardware explanation:**  
Data is read from a rotating disk by heads that do not move during reads/writes (moving takes very long).
- **Physics explanation:**  
**Nonexistent**, it seems!  
(A hardware designer might disagree?)

# Fetching data in parallel

- Parallel disk model with  $P$  disks:



- $P = \Theta(t^3)$  disks fit at distance  $\Theta(t)$ .
- Can simulate one disk with  $P$  read heads with small overhead. [Sanders et al. '00, '01]

# sI/O model

- A **sI/O** operation:  
Processor requests/writes **t** *arbitrary* words from/to external memory.
- The **s** stands for  
symmetric/scattered/super/...
- Equivalent to special case of parallel disk model with  $P=t$  and  $B=1$ .
- Physical realizability for  $t \rightarrow \infty$ : **Open.**

## Comparison to a PRAM

- sI/O model is similar to a CRCW PRAM with  $t$  processors.
- Difference 1: Computation is for free.
- Difference 2: Communication among processors is for free.

## Case study: Sorting

- Well studied in parallel disk model.
- Plugging in  $P=t$  and  $B=1$  gives:

$$\Theta\left(\frac{N}{t} \log_M(N/M)\right) \text{ sI/Os}$$

- *Slightly* less than than number of I/Os for  $B=t$ . Bottleneck is **temporal locality**.
- However, note that special cases such as *permuting* and *bucket sorting* now require only  $O(N/t)$  sI/Os.

## Case study: BFS

- I/O bottleneck in BFS is **spatial locality**.
- Use the standard internal memory implementation (using a buffered queue).
- The  $t$  first vertices in the queue can be tested for being already reached in  $1$  sI/O.
- Whenever  $t$  new vertices have been reached, their  $k$  neighbors can\* be retrieved in  $O(1+k/t)$  sI/Os.
- Total complexity:  $O(N/t+D)$  sI/Os, where  $D$  is the diameter of the graph.

# Is memory parallelism coming?

Some evidence:

- High performance external memory algorithms may utilize  $\sim 10$  disks.
- Hardware implementations of hashing use parallel lookups in several memory components.
- RAMBO memory has been built which may be regarded as looking up  $w$  bits in parallel in different memory components. Allows a constant time priority queue.

# Open questions

- Is the sI/O model worth studying?
  - What is the relation to the I/O model?
  - Is the notion of cache obliviousness useful for this model?
- Could we base lower bounds directly on the *laws of physics* rather than on a model? (Assume all computation takes place in a central processor.)

# Thank you

- Questions?
- Comments?
- Bursts of outrage?