Big Consequences of Little's Law

Larry Kaplan

April 26th, 2023

Salishan Conference on High Speed Computing

What is Little's Law?

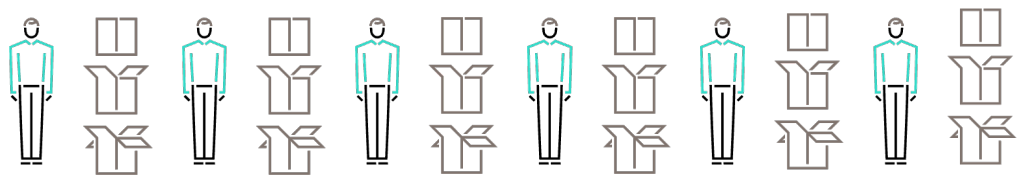
- Theorem in queuing theory for a system providing a service

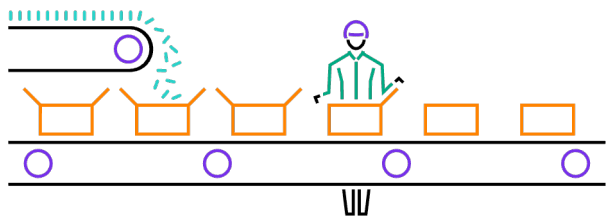
  average number of concurrent service requests =
  arrival rate of requests * time to service each request

- Expressed as

  L (# of requests) = λ (arrival rate) * W (wait time)

- Simple example
  - Can make 1 widget in 1 minute
  - Want to provide customers 6 widgets per minute
  - Strategies:
    - 6 separate widget makers
    - One 6-step production line with 10 seconds per step (after priming)
    - Etc.





"A proof for the queuing formula: L = λW", Little, J. D. C., Operations Research 9(3) 383–387, 1961

Little's Law in Computer Design

- Can be applied in many places that have bandwidth and latency constraints
  - Cache systems
  - Memory systems
  - Networks

Concurrency = Bandwidth * Latency

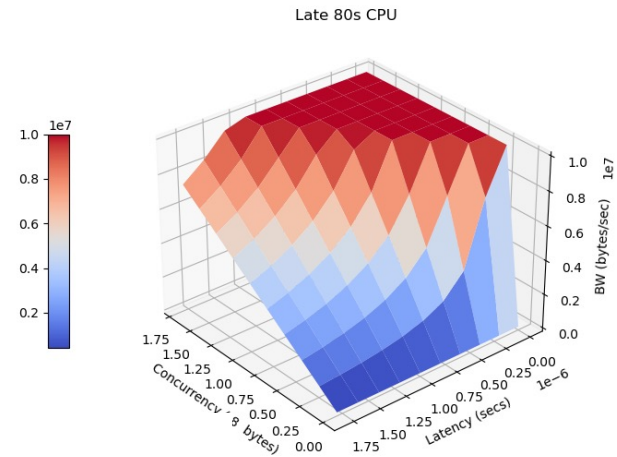
- Next slides show plots of Little's Law applied to memory systems and networks

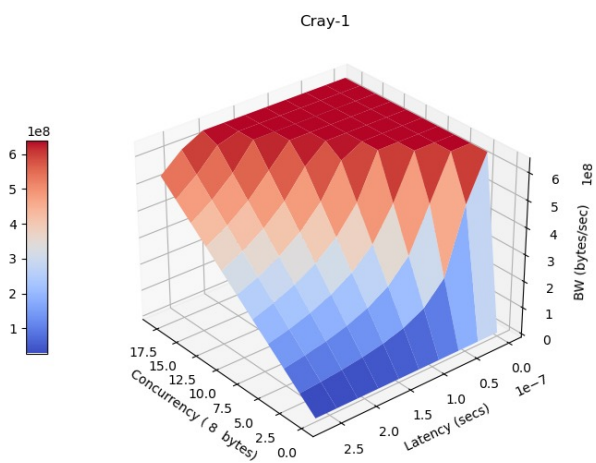
Given a) a design bandwidth ceiling for a memory or network and b) a latency to access that media
How much concurrency is required?

- Assume no bottlenecks elsewhere in system (not trivial)
- Latency measured from point of reference issue in processor

Little's Law Before 1990

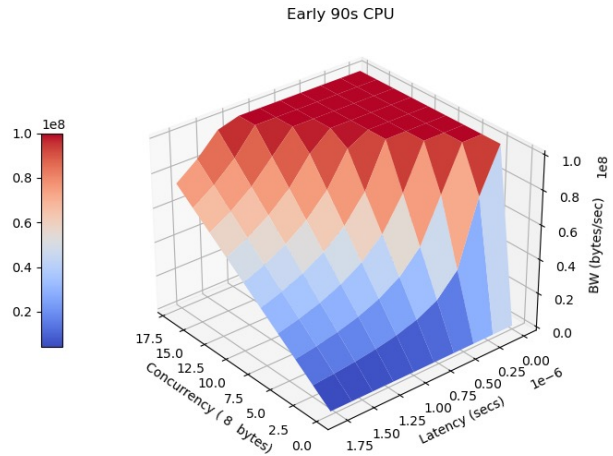
- Not much of an issue except in very high-end systems
  - Fully synchronous throughout
- Cray 1 – vectors and memory system pipelining provide required concurrency

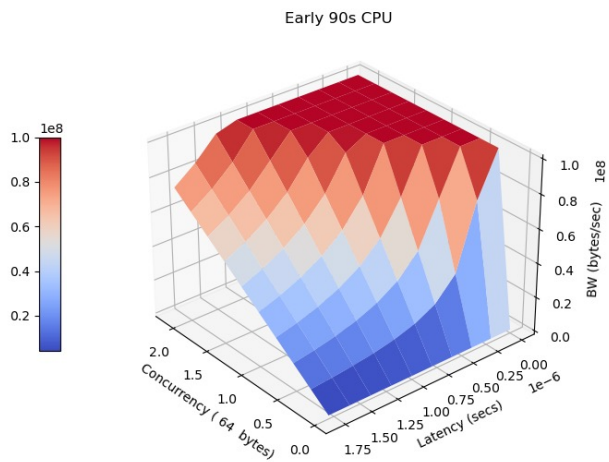




Little's Law Early 1990s

- Single word bandwidth*latency product starts going up
- Cache architectures become common
  - Intended to address disparity between CPU clock speeds and memory speeds
- Little's Law still mostly met due to increased access size (cacheline)
- Assumes good spatial locality (extra benefit for temporal locality)





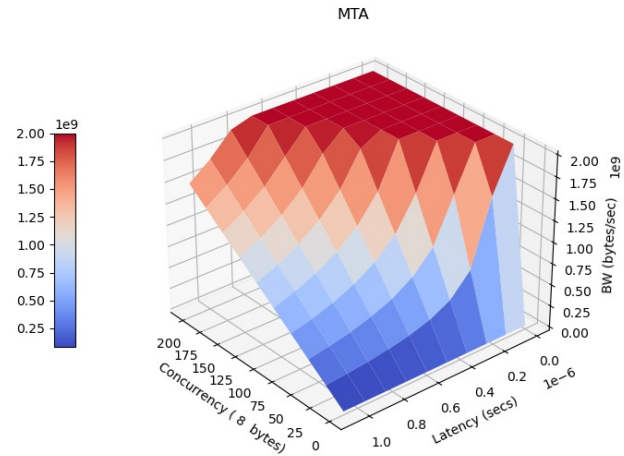
Challenges with Caches

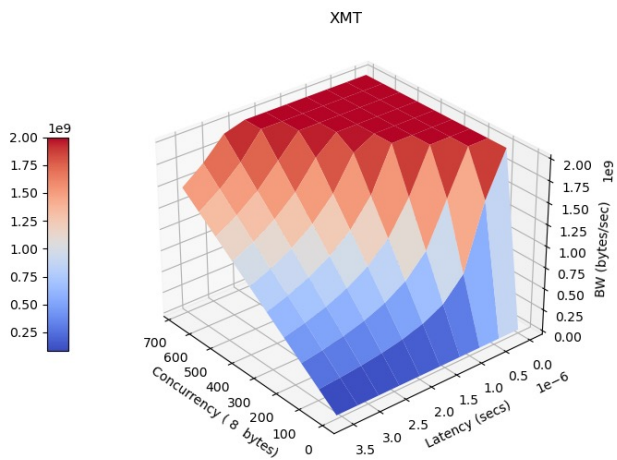
- Great when locality keeps miss rate low
- When miss rates get high, memory system becomes the bottleneck
  - Lack of temporal locality – data not reused before eviction (memory bandwidth used once)
  - Lack of spatial locality – only part of cache line used (throws away memory bandwidth)
- Both are algorithm dependent
  - Spatial locality can be sometimes be improved with packing/gathers
- Certain algorithms are “degenerate”, especially ”pointer chasing” (often part of graph algorithms)
  - The next word reference depends on full completion of the previous reference
  - Little or no cache locality (neither temporal nor spatial)
  - Clever coding can implement more than one “chasing stream” in a single thread on some architectures
    - Increases concurrency within a thread at the expense of increased register pressure

- CPU <-> memory connection not as wide as a cache line
  - Buffering and pipelining required resulting in increased memory system complexity

Tera MTA/Cray XMT

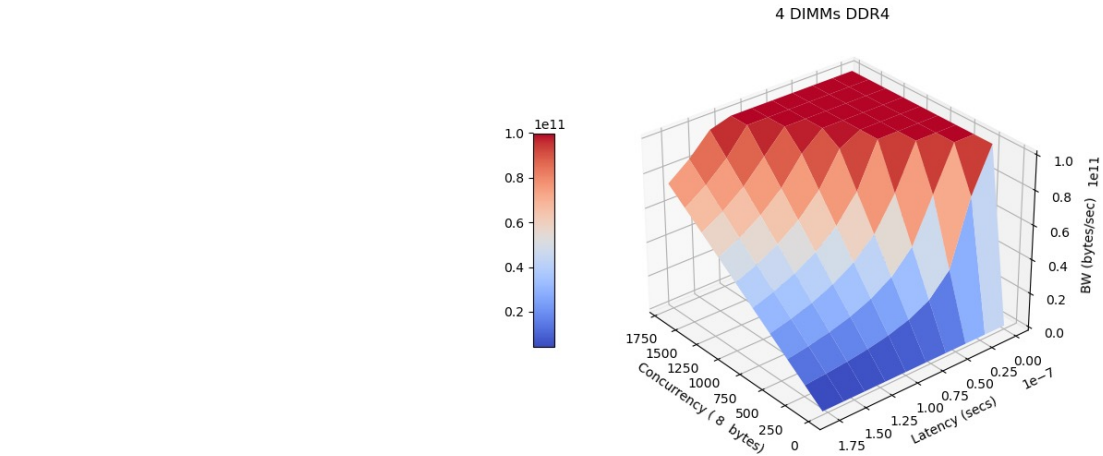
- Also in the 90s, MTA was designed based on Little's Law with a distributed uniform memory (UMA)
  - Latency tolerance using concurrency
- 128 HW threads, each with up to 8 outstanding memory references (typically 2-3), no data caches
  - Optimized for word accesses
- In 2000s XMT had > 3x latency, concurrency no longer always enough, locality became important again (NUMA)

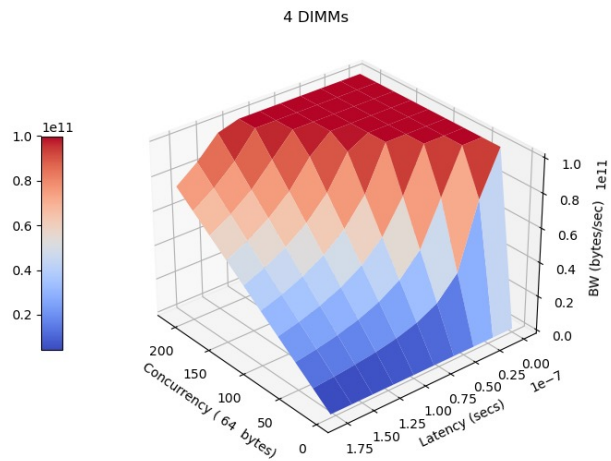




Modern DDR

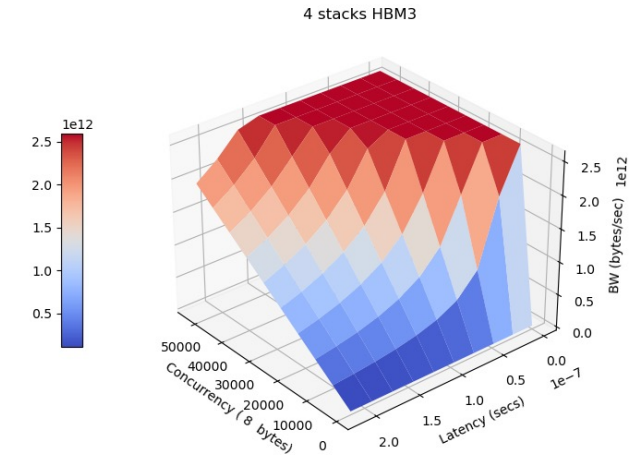
- With word references, a lot of concurrency is required
- Caches bring this down (typically by a factor of 8) but still challenging even with many-core CPUs
  - More concurrency is required

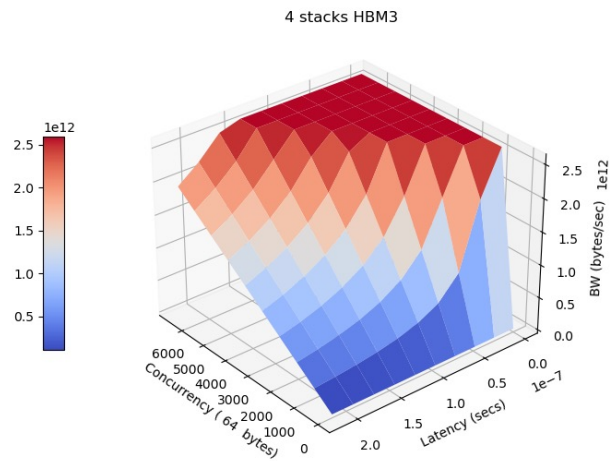




Challenge of HBM

- 25x bandwidth at slightly higher latency
- Where is all the concurrency going to come from?





Modern Concurrency Sources

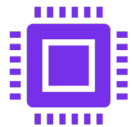
Hyperthreads

Best for pointer chasing?

Vectors (again)

Dense

“Gather” style vector references for sparse



GPUs/SIMT

Generate many latency tolerant memory references

Software

Reference density

Additional levels of parallelism

Modern Concurrency Challenges

Every level of the memory system, from load/store units, through all cache levels, across on-chip network, and at the memory controller must support sufficient concurrency

- Scatter/gather add complexity
- Significant “in-process” state to store and track
  - E.g., 0.5 MB of state for HBM3 at every level
- Low cache hit rates sometimes not properly provisioned with sufficient request concurrency to next level

Multi-channel/stack memory systems require good distribution strategies

- Naïve schemes subject to stride access conflicts

Significant algorithm dependence on generating many forms of concurrency

- Long vectors not always effective
- GPUs still struggle with some algorithms

Other Strategies to Address Consequences

Prefetch
(CPU or cache driven)

- More of a latency reduction technique (when successful)
- Can be a source of additional concurrency

Speculation

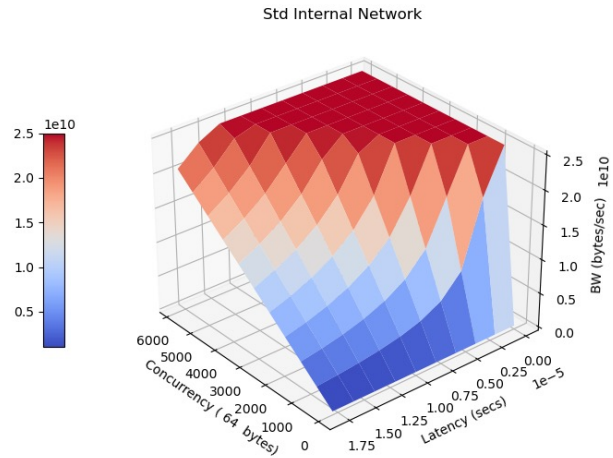
- Takes prefetching further into “possible” computational paths
- Far more likely to “waste” bandwidth than simple prefetch

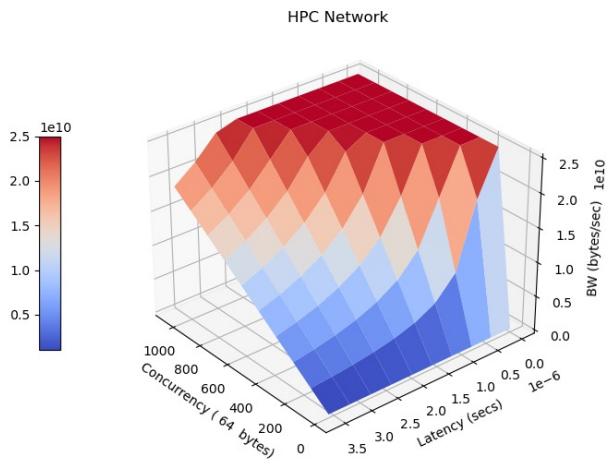
PIM and other techniques bringing memories closer to processors

- Reduced latency, reduces concurrency requirements

Modern Networks

- Network bound applications and communication phases face similar issues
  - Sparse matrix, pointer chasing, boundary exchanges
- HPC networks have similar bandwidth but typically lower latencies than standard ones





Myth Busters (and Related Challenges)

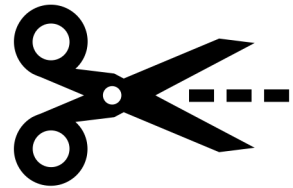
Little's Law Consequences

Myth 8	Everything Will Be Disaggregated!
Myth 12	All HPC Will Be Subsumed by the Clouds!
Myth 3	Extreme Specialization as Seen in Smartphones Will Push Supercomputers Beyond Moore's Law!

"Myths and Legends in High-Performance Computing", Satoshi Matsuoka, Jens Domke, Mohamed Wahib, Aleksandr Drozd, Torsten Hoefler, arXiv:2301.02432v1 [cs.DC] 6 Jan 2023

Myth Busters – Myth 8

Everything Will Be Disaggregated!



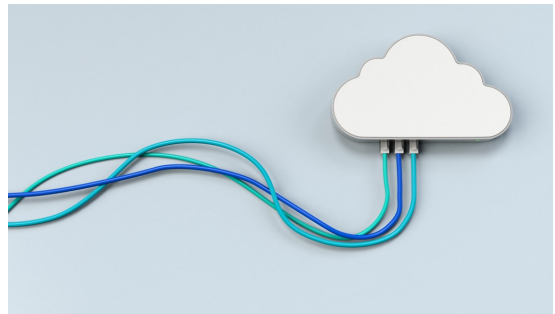
- Disaggregation adds latency
  - Comm/collective latency most important for “bulk-synchronous programming” (BSP)
  - Also increases concurrency requirements (close to 1000 for HPC networks)

- Pipelining also required to achieve bandwidth
  - Complicated buffering
  - Optics not amenable to buffering
  - Optical/electrical domain changes add yet more latency and complexity

- OK with disaggregation?
  - Large distributed, in-memory data structures
  - Loosely coupled applications and workloads

“Myths and Legends in High-Performance Computing”, Satoshi Matsuoka, Jens Domke, Mohamed Wahib, Aleksandr Drozd, Torsten Hoefler, arXiv:2301.02432v1 [cs.DC] 6 Jan 2023

Myth Busters – Myth 12
All HPC Will Be Subsumed by the Clouds!



- "Cloud" can mean many different things
  - Public vs private, access methodologies
  - Focus here on network type and data movement

- True HPC systems hosted in the cloud can provide the same functionality as on-premise
  - Business issues dominate in that case
    - E.g., public cloud vs. hosted private cloud, data storage/staging, elasticity, shared vs private resources
  - Not clear how common or cost-effective true HPC systems will ever be in public cloud

- Without low-latency HPC network, many applications are adversely affected
  - BSP suffers (see Myth 8)
  - Additional concurrency needed in communication for Little's Law
  - Will public cloud providers be sufficiently motivated to provide suitable systems and networks?

- Some applications are amendable to cloud, but data transfer/sharing still at issue

"Myths and Legends in High-Performance Computing", Satoshi Matsuoka, Jens Domke, Mohamed Wahib, Aleksandr Drozd, Torsten Hoefler, arXiv:2301.02432v1 [cs.DC] 6 Jan 2023

Myth Busters – Myth 3

Extreme Specialization as Seen in Smartphones Will Push Supercomputers Beyond Moore's Law!

- Asynchronous task-based programming likely necessary to address scheduling/load balancing efficiently
  - Also has the possibility of exposing additional degrees/levels of concurrency
- While such frameworks exist in HPC, they haven't caught on widely
  - Charm++, Legion, ParalleX/HPX, Chapel, OpenMP tasking, etc.
- Reasoning about this style of programming is harder than BSP

- However, heterogeneity is becoming a fact of life for some codes and workflows
  - Multi-physics, cogsim (coupled AI and simulation), etc.
- New specialized hardware is gaining traction

- Is increased concurrency another motivation to adopt such frameworks? Is it feasible?
  - Requires moving away from BSP and adopting other decomposition techniques
  - Weak vs Strong scaling issues
  - How wide can the task graphs be?
- Fantasy? Move from "time-based" to "event-based" simulation

"Myths and Legends in High-Performance Computing", Satoshi Matsuoka, Jens Domke, Mohamed Wahib, Aleksandr Drozd, Torsten Hoefler, arXiv:2301.02432v1 [cs.DC] 6 Jan 2023

Conclusion

- Little's Law is very relevant to computer design, both hardware and software

- Memory systems, cache hierarchies, and network interfaces must support sufficient concurrency
  - Requires complex buffering and pipelining

- Processor hardware must generate sufficient concurrency

- Some new hardware designs directly address Little's Law
  - Better vector units supporting effective long/sparse vectors (and other scatter/gather techniques)
  - Increased NUMA to reduce latency with sufficient bandwidth and concurrency
- But some seem to ignore it (insufficient concurrency)

- Algorithms must drive processor hardware sufficiently
  - Pointer chasing still difficult
  - Continuing improvements for other "sparse" reference patterns
  - Expose concurrency at multiple levels

Thank you