

# Utilizing Concurrency: A New Theory for Memory Wall

Xian-He Sun<sup>(✉)</sup> and Yu-Hang Liu

Illinois Institute of Technology, Chicago, USA  
{sun,yuhang.liu}@iit.edu

**Abstract.** In addition to locality, data access concurrency has emerged as a pillar factor of memory performance. In this research, we introduce a concurrency-aware solution, the memory Sluice Gate Theory, for solving the outstanding memory wall problem. Sluice gates are designed to control data transfer at each memory layer dynamically, and a global control algorithm, named layered performance matching, is developed to match the data transfer request/supply at each memory layer thus matching the overall performance between the CPU and memory system. Formal theoretical analyses are given to show, with sufficient data access concurrency and hardware support, the memory wall impact can be reduced to the minimum. Experimental testing is conducted which confirm the theoretical findings.

## 1 Introduction and Highlight

Memory wall problem refers to the relatively slow memory performance forming a wall between CPU and memory [1]. This wall causes CPUs to stall while waiting for data and slows down the speed of computing. The widely accepted solution for memory wall problem is the memory hierarchy approach. During the last thirty years, the design of the memory hierarchy has been enhanced to have more layers, larger caches, and built-in on-chip caches to match the increasingly large performance gap between computing and memory access. Besides the traditionally-focused locality, data access concurrency has become increasingly important, and can determine the performance of a memory system [2, 3].

Concurrency has been built into each layer of a memory hierarchy to support concurrent data access. However, a system is hard to reach the optimal locality and concurrency at the same time. Even it does, that does not mean it has reached the optimal system performance. Similarly, adding the optimizations of each memory layer of a memory hierarchy does not necessarily lead to the best system optimization. Locality and concurrency influence each other, within their layer and beyond their layer, and the influences are application dependent. These complicate the concurrency-aware data access optimization process.

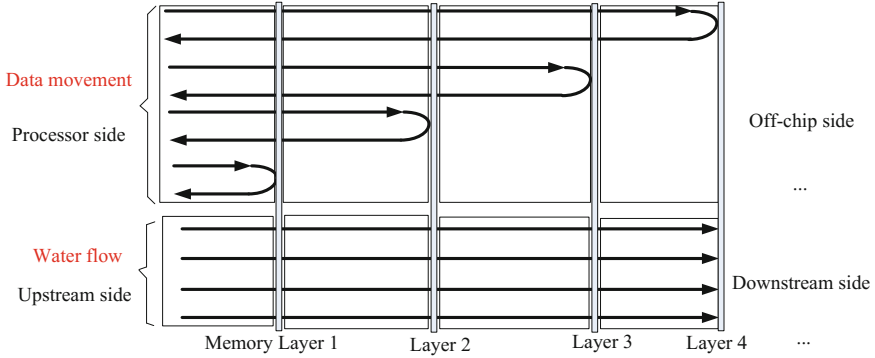
In this study, we propose a new theory, Sluice Gate Theory, to fully utilize memory hierarchy systems. Sluice Gate Theory claims that memory hierarchy is a designed sluice to transfer data to computing units, and through multi-level sluice gate control we can match data flow demand with supply. Therefore, we can reduce memory stall

time to the minimum under existing technologies, and provide a practical solution for the long-standing memory wall problem. Two techniques, the C-AMAT (Concurrent AMAT) model and the LPM (Layered Performance Matching) method, are developed to provide a constructive proof for Sluice Gate Theory.

C-AMAT serves as a gate calculator which finds a locality-concurrency balanced optimal configuration to match the data access requests and supplies at each layer of a memory hierarchy [2]. LPM controls the global memory system optimization and provides global control parameters to each memory layer [4]. Sluice Gate Theory provides a formal proof of the correctness of the LPM approach. That is, with sufficient data access and hardware concurrency, the LPM method can find a system configuration to match the demand with supply, whereas the matching will reduce the memory stall time to the minimum. Sluice Gate Theory utilizes the substantial memory concurrency that already exists at each layer of current memory systems to explore the combined effort of capacity, locality, and concurrency; and provides a constructive method for software and hardware co-design of memory systems. Only major theoretical results are presented in this paper. All the proofs can be found in [5], and the paper of C-AMAT [2] and LPM [4] are available online.

Sluice Gate Theory proves that through “matching” at each memory layer, the memory stall time can be reduced to the minimum. The terms “sluice” and “gate” are carefully chosen, implying data moves toward the computing unit in a specially designed, gate controlled data channel. Figure 1 illustrates data movement and the “sluice” and “gate”. The channel has stages with different devices (the registers, multi-level on-chip or off-chip caches, main memory, disk, and so on), has width in different forms (concurrency), and has speed in different measurements (bandwidth, frequency, latency). It is multi-staged to mask the performance difference between computing units and memory devices. At each stage, a “sluice gate” is placed to control the data movement. C-AMAT measures the supply rate and controls the “width” of the channel by increasing data access concurrency to meet the data access demand at each memory layer. This concurrency is not only for improving the data movement speed, but equally important for overlapping computing and data transfer. Data locality will increase the cache hits at the “gate” and, therefore, reduce the request at the next level of the memory hierarchy. The number of stages can be increased to improve concurrency, locality, and to adapt a new hardware device.

The LPM algorithm controls the matching process. It determines the data demand/supply matching threshold at each memory layer, and makes sure the thresholds can be reached through locality and concurrency optimizations. Due to the request and device differences at each stage, the sluice gates need to be locally controlled and adjusted to best fit the local demand. Since the performance at one memory layer will influence the performance of other memory layers, the performance matching of a memory system needs to be globally coordinated. Performance matching of a memory system is an uneasy task. Fortunately, the C-AMAT model and LPM algorithm have been developed for local calculation and global coordination, respectively. Jointly, C-AMAT and LPM provide a constructive proof of the Sluice Gate Theory.



**Fig. 1.** Compare between data access movement and water flow

## 2 The Theoretical Treatment of Memory Sluice Gate Theory

**Theorem 1 (Layered Performance Matching (LPM)):** *If a matching can be achieved at each memory layer for a given application for any matching threshold  $T > 0$  through optimization, then the LPM algorithm can find a performance matching for the application.*

With the LPM theorem, we now analyze the assumptions of the LPM theorem.

**Theorem 2 (Data Concurrency):** *If an application has sufficient hit concurrency and has sufficient pure miss concurrency or sufficiently low pure miss rate or pure miss penalty at layer  $L_i$ , then at memory layer  $L_i$ , we can find a performance matching for any matching threshold  $T_i > 0$ .*

All the optimization parameters used in Data Concurrency Theorem, hit concurrency, pure miss concurrency, pure miss rate, and pure miss penalty are data access concurrency parameters introduced by C-AMAT [2]. They can be optimized through increasing software and hardware concurrency. They do not depend on the memory device hardware peak performance. In other words, the concurrency theorem says through concurrency improvement we can find a match at memory layer  $L_i$ . The theorem shows the great potential of data access concurrency.

Based on the Data Concurrency and the LPM Theorem, the following result shows that we can remove the memory wall effect through increasing data concurrency.

**Theorem 3 (Concurrency Match):** *If an application has sufficient hit concurrency and has sufficient pure miss concurrency or sufficiently low pure miss rate or pure miss penalty at each memory layer, then the LPM algorithm can find a performance matching for the application for any matching threshold  $T_1 > 0$ .*

The proof of the Concurrency Match Theorem has only used the concurrency parameters. The following theorem shows the contribution of data locality in performance matching.

**Theorem 4 (Data Locality):** *Increasing data locality at memory layer  $j$  ( $1 \leq j \leq i$ ), will decrease the data access request rate at the memory layer  $L_{i+1}$ .*

From Data Concurrency Theorem and Data Locality Theorem, we can see data concurrency and data locality playing different roles in the performance matching process. Data concurrency improves the supply in a memory performance matching, and data locality reduces the request in memory performance matching. They are both vital in memory performance matching.

Recall the impact of the memory wall problem is the large ratio of memory stall time compared to the total application runtime. Therefore, we can claim that the memory wall effect is negligible small if memory stall time is less than 1% of the application's pure execution time (we think 1% is small enough, but it can be  $x\%$  for any  $x > 0$ ). With this one percent definition, we have the final result.

**Theorem 5 (Sluice Gate):** *If a memory system can match an application's data access requirement for any matching threshold  $T_1 > 0$ , then this memory system has removed the memory wall effect for this application.*

The Sluice Gate Theorem is of great significance. It claims that the memory wall impact can be reduced to the minimum and to be practically eliminated through data access concurrency, on conventional memory hierarchy architectures. For a long time, the memory wall problem has been the wall standing on the road of improving computing system performance. It has been believed that the memory wall problem only can be solved through technology advancements of memory devices. The Sluice Gate Theorem gives an alternative approach via data concurrency.

The performance match can be found as stated in Data Concurrency Theorem is in a theoretical sense. Theoretically achievable does not mean we can achieve it in today's engineering practice, but through engineering effort we may achieve it someday. While we may not have sufficient data access and dynamic hardware concurrency in practice, Sluice Gate Theory gives a direction of software/hardware co-design and optimization to reduce memory stall time to the minimum.

### 3 Experimental Results and Conclusion

A detailed CPU model and the DRAMSim2 module in the GEM5 simulator were adopted to achieve accurate simulation results. We have conducted several case studies, and only show one, the Multiple Dimension Exploration case study, here in.

Under the five configurations A to E, Table 1 shows the corresponding average LPMRs (LPM Ratios) of the 410.bwaves benchmark in the SPEC CPU 2006 benchmark suit. We use the LPM algorithm [4] to find an optimal architecture match for the given software implementation. The goal of the optimization is to keep the memory stall time per instruction within 1% of  $CPI_{exe}$ , where the  $CPI_{exe}$  is 0.261 cycles per instruction on average. The calculated matching thresholds,  $T_1$  and  $T_2$ , for L1 and L2 cache of the 410.bwaves benchmark are 1.52 and 2.14, respectively. Table 1 shows under Configuration A, the LPMRs are higher than the threshold values of  $T_1$  and  $T_2$ , so that the optimizations are carried in both layers at the same time. To increase

concurrency, we doubled the IW and ROB size, transformed the architecture from configuration A to configuration B in Table 1. However, the mismatches are still higher than their thresholds. Then we continue the optimization process and transform the configuration B to configuration C, and then to D. Configuration D meet the “1%” requirement. As an optional step, we continue to check if hardware is overprovided. We do a fine tune to reduce possible hardware overprovision to achieve cost efficiency, which leads to the final configuration E.

**Table 1.** LPMRs under five machine configurations

Configuration		A	B	C	D	E
Sluice width	Pipeline issue width	4	4	6	8	8
	IW size	32	64	64	128	96
	ROB size	32	64	64	128	96
	L <sub>1</sub> cache port number	1	1	2	4	4
	MSHR numbers	4	8	16	16	16
	L <sub>2</sub> cache interleaving	4	8	8	8	8
Mismatching degree	LPMR <sub>1</sub>	8.1	6.2	2.1	1.2	1.4
	LPMR <sub>2</sub>	9.6	9.3	3.1	1.6	1.9

Please notice with the original configuration A, the memory stall time is 0.396 cycles per instruction, which contributes more than 60% of the total execution time (0.653 cycles per instruction). With the configuration E, the final memory stall time is less than 1% of the pure execution time (which is less than 0.4% of the original total execution time). Therefore, the memory system performance speedup is greater than 150. The performance improvement is huge.

Sluice Gate Theory provides a system approach to solve the long-standing memory wall problem. Its correctness is verified with rigorous mathematical proofs, and its practical applicability is supported with its associated C-AMAT model and LPM method for performance measurement and optimization. Sluice Gate Theory utilizes existing data concurrency and optimizes the combined performance of data locality and concurrency to reduce the overall memory stall time. It is powerful and imperative for the advancement of modern memory systems. Sluice Gate Theory is based on data concurrency. It calls for the rethinking from a data centric view. It calls for the development of compiler technologies to utilize data access concurrency and to develop concurrency-aware locality optimizations, and provides a guideline for such optimization and utilization.

## References

1. Wulf, W.A., McKee, S.A.: Hitting the memory wall: implications of the obvious. ACM SIGARCH Comput. Archit. News **23**, 20–24 (1995)
2. Sun, X.H., Wang, D.: Concurrent average memory access time. IEEE Comput. **47**(5), 74–80 (2014)

3. Chou, Y., Fahs, B., Abraham, S.: Microarchitecture optimizations for memory-level parallelism. In: Proceedings of 31st International Symposium on Computer Architecture, June 2004
4. Liu, Y.H., Sun, X.H.: LPM: concurrency-driven layered performance matching. In: 44th International Conference on Parallel Processing (ICPP). IEEE (2015)
5. Sun, X.-H., Liu, Y.-H.: Sluice gate theory: have we found a solution for memory wall?. Illinois Institute of Technology Technical report (IIT/CS-SCS-2016-01) (2016). Full paper is available upon request

Languages and Compilers for Parallel Computing  
29th International Workshop, LCPC 2016, Rochester,  
NY, USA, September 28-30, 2016, Revised Papers  
Ding, C.; Criswell, J.; Wu, P. (Eds.)  
2017, XI, 348 p. 137 illus., Softcover  
ISBN: 978-3-319-52708-6