How to Stop Under-Utilization and Love Multicores

Anastasia Ailamaki[#] Erietta Liarou[†] Pınar Tözün[#] Danica Porobic[#] Iraklis Psaroudakis[#] [*]

[#]*École Polytechnique Fédérale de Lausanne*
[†]*Harvard University*
[*]*SAP AG*

***Abstract*—Hardware trends oblige software to overcome three major challenges against systems scalability: (1) taking advantage of the implicit/vertical parallelism within a core that is enabled through the aggressive micro-architectural features, (2) exploiting the explicit/horizontal parallelism provided by multicores, and (3) achieving predictively efficient execution despite the variability in communication latencies among cores on multisocket multicores. In this three hour tutorial, we shed light on the above three challenges and survey recent proposals to alleviate them.**

The first part of the tutorial describes the instruction- and data-level parallelism opportunities in a core coming from the hardware and software side. In addition, it examines the sources of under-utilization in a modern processor and presents insights and hardware/software techniques to better exploit the micro-architectural resources of a processor by improving cache locality at the right level of the memory hierarchy. The second part focuses on the scalability bottlenecks of database applications at the level of multicore and multisocket multicore architectures. It first presents a systematic way of eliminating such bottlenecks in online transaction processing workloads, which is based on minimizing unbounded communication, and shows several techniques that minimize bottlenecks in major components of database management systems. Then, it demonstrates the data and work sharing opportunities for analytical workloads, and reviews advanced scheduling mechanisms that are aware of non-uniform memory accesses and alleviate bandwidth saturation.

I. Introduction

Length: 3 hours

Target Audience: Researchers and developers in the field of data management systems who are non-experts on modern hardware and the challenges that emerging hardware poses on high-performance transaction and query processing. Also, PhD students who are interested in learning more about the underlying hardware and seeking a challenging and high-impact research topic on data management systems on modern hardware.

Related Previous Tutorials: This tutorial expands our SIGMOD 2014 tutorial by 20% [1]. In addition to what was presented there, here we plan to discuss in detail recent on new storage devices such as non-volatile memory, dynamic query compilation, and specialized hardware for database operations. Furthermore, the first half of the second part of this tutorial, related to scaling up OLTP on multicores, borrows some content from our VLDB 2013 tutorial titled *Toward Scalable Transaction Processing - Evolution of Shore-MT* [2]. However, this constitutes less than 20% of this tutorial. Slides for both of these tutorials can be found at https://sites.google.com/site/shoremt/presentations.

II. Implicit/Vertical Parallelism

In step with Moore's law [3], processor technology has gone through major advancements over the years. Before the last decade hardware vendors mainly innovated on implicit parallelism within a core boosting the performance of a single thread. They either kept clocking the processors at higher frequencies or designing aggressive micro-architectural features (e.g., long execution pipelines, super-scalar execution, out-of-order execution, branch prediction, vector processing, etc. [4]) that increase the complexity of a processor. However, taking advantage of such features is never straightforward for the complex data management applications [5], [6], mainly due to the low instruction level parallelism they exhibit. These applications usually require fundamental algorithmic changes in order to really exploit both data and instruction level parallelism opportunities that exist on modern processors [7], [8], [9].

The algorithmic changes that take into account the micro-architectural features of a core are only one part of the solution. One also needs to account for the memory hierarchy on the machines being used. Recent studies analyzing the micro-architectural behavior of typical data management workloads on modern hardware emphasize that more than half of the execution time goes to memory stalls when running data intensive workloads [10]. As a result, on processors that have the ability to execute four instructions per cycle (IPC), which is common for modern commodity hardware, data intensive workloads, especially transaction processing, achieve around one instruction per cycle [11]. Such under-utilization of micro-architectural features is a great waste of hardware resources.

Several proposals have been made to reduce memory stalls through improving instruction and data locality to increase cache hit rates. For data, these range from cache-conscious data structures and algorithms [12] to sophisticated data partitioning and thread scheduling [13]. For instructions, they range from compilation optimizations [14], and advanced prefetching [15], to computation spreading [16], [17] and transaction batching for instructions [18], [19]. In addition, several recent proposals opt for hardware specialization for some of the database operations ([20], [21], [22]).

In this part of the tutorial, we first give an overview of the instruction and data parallelism opportunities in a core, as well as the typical memory hierarchy of a server processor today. Then, we illustrate the strengths and weaknesses of the techniques that aim to better utilize micro-architectural

resources of a core with examples from recent work while presenting the key insights behind each of them.

III. Explicit/Horizontal Parallelism

Since the beginning of this decade, power draw and heat dissipation prevent processor vendors from relying on rising clock frequencies or more aggressive micro-architectural techniques for higher performance. Instead, they add more processing cores or hardware contexts on a single processor to enable exponentially increasing opportunities for parallelism [23]. Exploiting this parallelism is crucial for utilizing the available architectural resources and enabling faster software. However, designing scalable systems that can take advantage of the underlying parallelism remains a challenge. In traditional high performance transaction processing, the inherent communication leads to scalability bottlenecks on today's multicore and multisocket hardware. Even systems that scale very well on one generation of multicores might fail to scale-up on the next generation. On the other hand, in traditional online analytical processing, the database operators that were designed for unicore processors fail to exploit the abundant parallelism offered by modern hardware.

Servers with multiple processors and non-uniform memory access (NUMA) design present additional challenge do data management systems, many of which were designed with implicit assumptions that core-to-core communication latencies and core-to-memory access latencies are constant regardless of location. However, today for the first time we have *Islands*, i.e., groups of cores that communicate fast among themselves and slower with other groups. Currently, an Island is represented by a processor socket but soon, with dozens of cores on the same socket, we expect that Islands will form within a chip. Additionally, memory is accessed through memory controllers of individual processors. In this setting, memory access times vary greatly depending on several factors including latency to access remote memory and contention for the memory hierarchy such as the shared last level caches, the memory controllers, and the interconnect bandwidth.

Abundant parallelism and non-uniformity in communication present different challenges to transaction and analytical workloads. The main challenge for transaction processing is communication. In this part of the tutorial, we initially teach a methodology for scaling up transaction processing systems on multicore hardware. More specifically, we identify three types of communication in a typical transaction processing system: *unbounded, fixed, and cooperative* [24]. We demonstrate that the key to achieve scalability on modern hardware, especially for transaction processing systems, but also for any system that has similar communication patterns, depends on avoiding the unbounded communication points or downgrading them into fixed or cooperative ones. We show how effective this methodology is in practice by surveying related proposals from recent work (e.g., [25], [26], [27], [28], [29], [30]).

Non-uniform communication latencies make it appealing to regard multisocket as a distributed system and deploy multiple nodes in a shared-nothing configuration [26], [28]. While this approach works great for perfectly partitionable workloads, it is very sensitive to distributed transactions and the workload skew. At the same time, hardware-oblivious shared-everything systems suffer from non-uniform latencies that amplify bottlenecks in the critical path [31]. In order to achieve scalability on multisockets one needs to make the system aware of the hardware topology and dynamically adapt to workload and hardware [13].

On the other hand, traditional online analytical processing workloads are formed of scan-heavy, complex, ad-hoc queries that do not suffer from the unbounded communication as in transaction processing. Analytical workloads are still concerned with the variability of latency, but also with avoiding saturating resources such as memory bandwidth. In many analytical workloads that exhibit similarity across the query mix, sharing techniques can be employed to avoid redundant work and re-use data in order to better utilize resources and decrease contention. We survey recent techniques that aim at exploiting work and data sharing opportunities among the concurrent queries (e.g., [32], [33], [34], [35]).

Furthermore, another important aspect of analytical workloads, in comparison to transaction processing, is intra-query parallelism. Typical database operators, such as joins, scans, etc., are mainly optimized for single threaded execution. Therefore, they fail to exploit intra-query parallelism and cannot utilize several cores naïvely. We survey recent parallelized analytical algorithms on modern non-uniform multisocket multicore architectures [36], [37], [38], [39].

Finally, in order to optimize performance on non-uniform platforms, the execution engine needs to tackle two main challenges for a mix of multiple queries: (a) employing a scheduling strategy for assigning multiple concurrent threads to cores in order to minimize remote memory accesses while avoiding contention on the memory hierarchy, and (b) dynamically deciding on the data placement in order to minimize the total memory access time of the workload. The two problems are not orthogonal, as data placement can affect scheduling decisions, while scheduling strategies need to take into account data placement. We review the requirements and recent techniques for highly concurrent NUMA-aware scheduling for analytics, which take into consideration data locality, parallelism, and resource allocation (e.g., [40], [41], [42], [43]).

IV. Tutorial Outline

- INTRODUCTION AND OVERVIEW (10 minutes)
  - Tutorial overview: goal, audience, and schedule
  - Hardware trends
  - Problem statement:
    - dimensions of parallelism
    - challenges traditional data management systems face on modern hardware
- WHAT HAPPENS IN A CORE (40 minutes)
  - Instruction Level Parallelism
  - Data Level Parallelism and SIMD

- Simultaneous Multithreading
- MINIMIZING MEMORY STALLS (40 minutes)
  - Results from recent workload characterization studies
  - Techniques to improve instruction and data cache locality
  - Toward specialized hardware
- SCALING UP OLTP (40 minutes)
  - Communication types in transaction processing
  - Recent work on scaling up OLTP on modern hardware by identifying and eliminating the various communication types
  - Quantifying the impact of non-uniform communication on OLTP performance
  - Dynamically adjusting to the hardware topology and workload characteristics on NUMA hardware
- SCALING UP OLAP (40 minutes)
  - Memory access bottlenecks in multisocket multicore architectures
  - Sharing data and work across concurrent analytical queries
  - NUMA-aware parallel analytical algorithms
  - NUMA-aware scheduling for highly concurrent analytical workloads
- CONCLUSIONS AND FUTURE DIRECTIONS (10 minutes)

V. Biography

Anastasia Ailamaki is a Professor of Computer Sciences at École polytechnique Fédérale de Lausanne (EPFL) in Switzerland. Her research interests are in the broad area of database systems and applications, with emphasis on database system behavior on modern processor hardware and disks. Her projects aim at building systems to strengthen the interaction between the database software and the underlying hardware and I/O devices. She is also working on automated schema design and computational database support for scientific applications, as well as storage device modeling, performance prediction, and internet query caching. Anastasia has received an ERC Consolidator grant (2013), a Finmeccanica endowed chair from the Computer Science Department at Carnegie Mellon (2007), a European Young Investigator Award from the European Science Foundation (2007), an Alfred P. Sloan Research Fellowship (2005), seven best-paper awards at top conferences (2001-2011), and an NSF CAREER award (2002). She earned her Ph.D. in Computer Science from the University of Wisconsin-Madison in 2000. She is a member of IEEE and ACM, and has also been a CRA-W mentor. She also serves at the Global Agenda Council for Data, Society and Development of the World Economic Forum.

Erietta Liarou received her PhD in Computer Science from the University of Amsterdam in 2013. Her primary research interests include database architectures, transaction processing on modern hardware, stream processing, distributed query processing, and data analytics with emphasis on very large data management. She is a visiting researcher at the Data Systems Laboratory at Harvard SEAS, and she has also been with the Data Intensive Applications and Systems (DIAS) lab at EPFL, the System S group at IBM T. J. Watson Research Center and the Intelligent Systems Laboratory at Technical University of Crete.

Pınar Tözün has received her PhD in Computer Science from École polytechnique Fédérale de Lausanne (EPFL) in 2014 under the supervision of Prof. Anastasia Ailamaki in Data-Intensive Applications and Systems (DIAS) Laboratory. Her research focuses on scalability and efficiency of transaction processing systems on modern hardware. She received her BSc degree in Computer Engineering department of Koç University in 2009.

Danica Porobic is a fifth year PhD student at École polytechnique Fédérale de Lausanne (EPFL) working under the supervision of Prof. Anastasia Ailamaki in Data-Intensive Applications and Systems (DIAS) Laboratory. Her research focuses on designing scalable transaction processing systems for non-uniform hardware. She has graduated top of her class with MSc and BSc in Informatics from University of Novi Sad and has worked at Oracle Labs and Microsoft SQL Server.

Iraklis Psaroudakis is a fourth year PhD student at École polytechnique Fédérale de Lausanne (EPFL) working under the supervision of Prof. Anastasia Ailamaki in Data-Intensive Applications and Systems (DIAS) Laboratory. His research focuses on scheduling highly concurrent analytical workloads and he also co-operates with the SAP HANA database team. He has received his diploma from the School of Electrical and Computer Engineering of the National Technical University of Athens.

References

[1] A. Ailamaki, E. Liarou, P. Tözün, D. Porobic, and I. Psaroudakis, "How to Stop Underutilization and Love Multicores," in *SIGMOD*, 2014, pp. 189–192.

[2] A. Ailamaki, R. Johnson, I. Pandis, and P. Tözün, "Toward Scalable Transaction Processing: Evolution of Shore-MT," *PVLDB*, vol. 6, no. 11, pp. 1192–1193, 2013.

[3] G. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 6, 1965.

[4] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[5] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, "DBMSs on a modern processor: Where does time go?" in *VLDB*, 1999, pp. 266–277.

[6] N. Hardavellas, I. Pandis, R. Johnson, N. Mancheril, A. Ailamaki, and B. Falsafi, "Database servers on chip multiprocessors: Limitations and opportunities," in *CIDR*, 2007, pp. 79–87.

[7] C. Kim, T. Kaldewey, V. W. Lee, E. Sedlar, A. D. Nguyen, N. Satish, J. Chhugani, A. Di Blas, and P. Dubey, "Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-core CPUs," *VLDBJ*, vol. 2, no. 2, pp. 1378–1389, 2009.

[8] V. Raman, G. Swart, L. Qiao, F. Reiss, V. Dialani, D. Kossmann, I. Narang, and R. Sidle, "Constant-time query processing," in *ICDE*, 2008, pp. 60–69.

[9] K. A. Ross, "Selection Conditions in Main Memory," *ACM TODS*, vol. 29, no. 1, pp. 132–161, 2004.

[10] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware," in *ASPLOS*, 2012, pp. 37–48.

[11] P. Tözün, I. Pandis, C. Kaynak, D. Jevdjic, and A. Ailamaki, "From A to E: Analyzing TPC's OLTP Benchmarks – The obsolete, the ubiquitous, the unexplored," in *EDBT*, 2013, pp. 17–28.

[12] S. Chen, P. B. Gibbons, T. C. Mowry, and G. Valentin, "Fractal Prefetching B+-Trees: Optimizing Both Cache and Disk Performance," in *SIGMOD*, 2002, pp. 157–168.

[13] D. Porobic, E. Liarou, P. Tözün, and A. Ailamaki, "ATraPos: Adaptive Transaction Processing on Hardware Islands," in *ICDE*, 2014.

[14] A. Ramirez, L. A. Barroso, K. Gharachorloo, R. Cohn, J. Larriba-Pey, P. G. Lowney, and M. Valero, "Code Layout Optimizations for Transaction Processing Workloads," in *ISCA*, 2001, pp. 155–164.

[15] M. Ferdman, C. Kaynak, and B. Falsafi, "Proactive Instruction Fetch," in *MICRO*, 2011, pp. 152–162.

[16] I. Atta, P. Tözün, A. Ailamaki, and A. Moshovos, "SLICC: Self-Assembly of Instruction Cache Collectives for OLTP Workloads," in *MICRO*, 2012, pp. 188–198.

[17] K. Chakraborty, P. M. Wells, and G. S. Sohi, "Computation Spreading: Employing Hardware Migration to Specialize CMP Cores On-the-fly," in *ASPLOS*, 2006, pp. 283–292.

[18] I. Atta, P. Tözün, X. Tong, A. Ailamaki, and A. Moshovos, "STREX: Boosting Instruction Cache Reuse in OLTP Workloads through Stratified Transaction Execution," in *ISCA*, 2013, pp. 273–284.

[19] S. Harizopoulos and A. Ailamaki, "STEPS Towards Cache-Resident Transaction Processing," in *VLDB*, 2004, pp. 660–671.

[20] R. Johnson and I. Pandis, "The bionic dbms is coming, but what will it look like?" in *CIDR*, 2013.

[21] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the Walkers: Accelerating Index Traversals for In-memory Databases," in *MICRO*, 2013, pp. 468–479.

[22] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, "Q100: The Architecture and Design of a Database Processing Unit," in *ASPLOS*, 2014, pp. 255–268.

[23] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The Case for a Single-Chip Multiprocessor," in *ASPLOS*, 1996, pp. 2–11.

[24] R. Johnson, I. Pandis, and A. Ailamaki, "Eliminating unscalable communication in transaction processing," *VLDBJ*, vol. 23, no. 1, pp. 1–23, 2014.

[25] C. Diaconu, C. Freedman, E. Ismert, P.-A. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling, "Hekaton: SQL Server's Memory-optimized OLTP Engine," in *SIGMOD*, 2013, pp. 1243–1254.

[26] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots," in *ICDE*, 2011, pp. 195–206.

[27] I. Pandis, P. Tözün, R. Johnson, and A. Ailamaki, "PLP: Page Latch-free Shared-everything OLTP," *PVLDB*, vol. 4, no. 10, pp. 610–621, 2011.

[28] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The End of an Architectural Era: (It's Time for a Complete Rewrite)," in *VLDB*, 2007, pp. 1150–1160.

[29] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: Fast Distributed Transactions for Partitioned Database Systems," in *SIGMOD*, 2012, pp. 1–12.

[30] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy Transactions in Multicore In-memory Databases," in *SOSP*, 2013, pp. 18–32.

[31] D. Porobic, I. Pandis, M. Branco, P. Tözün, and A. Ailamaki, "OLTP on Hardware Islands," *PVLDB*, vol. 5, no. 11, pp. 1447–1458, 2012.

[32] G. Candea, N. Polyzotis, and R. Vingralek, "A Scalable, Predictable Join Operator for Highly Concurrent Data Warehouses," *PVLDB*, vol. 2, no. 1, pp. 277–288, 2009.

[33] G. Giannikis, G. Alonso, and D. Kossmann, "SharedDB: Killing One Thousand Queries with One Stone," *PVLDB*, vol. 5, no. 6, pp. 526–537, 2012.

[34] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki, "QPipe: A Simultaneously Pipelined Relational Query Engine," in *SIGMOD*, 2005, pp. 383–394.

[35] I. Psaroudakis, M. Athanassoulis, and A. Ailamaki, "Sharing Data and Work Across Concurrent Analytical Queries," *PVLDB*, vol. 6, no. 9, pp. 637–648, 2013.

[36] L. Qiao, V. Raman, F. Reiss, P. J. Haas, and G. M. Lohman, "Main-memory Scan Sharing for Multi-core CPUs," *VLDB*, 2008.

[37] Y. Li, I. Pandis, R. Mueller, V. Raman, and G. Lohman, "NUMA-aware Algorithms: The Case of Data Shuffling," in *CIDR*, 2013.

[38] M.-C. Albutiu, A. Kemper, and T. Neumann, "Massively Parallel Sort-merge Joins in Main Memory Multi-core Database Systems," *PVLDB*, vol. 5, no. 10, pp. 1064–1075, 2012.

[39] C. Balkesen, G. Alonso, J. Teubner, and M. T. Ozsu, "Multi-Core, Main-Memory Joins: Sort vs. Hash Revisited," *PVLDB*, vol. 7, no. 1, 2014.

[40] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quma, and M. Roth, "Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems," in *ASPLOS*, 2013, pp. 381–394.

[41] I. Psaroudakis, T. Scheuer, N. May, and A. Ailamaki, "Task Scheduling for Highly Concurrent Analytical and Transactional Main-Memory Workloads," *ADMS*, 2013.

[42] J. Dees and P. Sanders, "Efficient many-core query execution in main memory column-stores," in *IEEE 29th International Conference on Data Engineering (ICDE)*, ser. ICDE '13, 2013, pp. 350–361.

[43] V. Leis, P. Boncz, A. Kemper, and T. Neumann, "Morsel-driven parallelism: A numa-aware query evaluation framework for the many-core age," in *SIGMOD*, 2014.