# In-Memory Data Analytics on Coupled Architectures

Bingsheng He

## 1 Background on Coupled Architectures

Modern CPU has evolved into powerful processor with multi-core and high frequency processing capabilities. However, due to the manufacturing limitations and thermal issues, it becomes increasingly difficulty to further improve the computing capability of CPU. Moreover, the increasing complex applications require processors to be able to process workloads with varying computing complexity and pattern. Accelerators are designed to bridge the gap between CPU and requirements in reality. Graphics Processing Unit (GPU) is specifically designed for manipulating graphic and image processing. Field Programmable Gate Array (FPGA) is specialized with configurable capability to deliver combinational functions customized by users.

### 1.1 Why coupled

Since the emerging of accelerators, they are designed to serve specific purposes. In a long time, heterogeneous systems equipped with CPU and dedicated accelerators have been widely adopted in various domains. On the one hand, heterogeneous systems can be provide additional computing capability beyond CPU-based systems. On the other hand, accelerators in such systems can process the parts of complex workloads with specialized requirements faster than CPU. As video stream processing involves large amount of computations and the video stream usually has a limited lifetime, resulting that it is not well suited to CPU as caches cannot be fully utilized. On the contrary, accelerators like GPU work efficiently on such workloads as it has greatly simplified memory hierarchy and control logics. Instead, more hardware resources are devoted to light-weight and massively parallel Arithmetic Logic Units (ALUs) that is suitable for such workloads. More generally, transistors in CPU are dedicated to branch prediction, out of order logic and caching to reduce the latency to memory, thus the performance of each thread is maximized. However, most transistors of GPU are concentrated on ALUs and registers, thus each thread's state can be stored, allowing faster context switching between threads to cover, rather than to reduce, the latency to memory.

Though integrating accelerators into the system has eased the burden of CPU to process complex workloads much faster and more efficiently, additional overhead is incurred, hindering its wider adoption to obtain further performance improvements.

Firstly, most traditional systems incorporate CPU and accelerators connected via PCI-e to transfer data between them. Though each processor has enough hardware space so that they can be designed powerful enough, the data transfer overhead and coordination between CPU and accelerators make it inefficient to run applications on the system. On the one hand, the bandwidth of PCI-e is limited (e.g., 4 8 GB/sec). On the other hand, the performance of PCI-e seriously downgrades in cases where the size of data blocks is small.

Secondly, due to the data transfer overhead, workloads are usually offloaded to accelerators for processing as a whole. Once computations end, the entire results are transferred back to the CPU for post-processing. Such naive workload scheduling is PCi-e favoured. However, it always results in imbalance between CPU and accelerators. The CPU is always idle while the accelerator is busy with compute-intensive workloads.

Thirdly, traditional applications on CPU+accelerator platforms do not consider the dynamics in complex applications due to hardware constraints. Complex applications always involve various kernels with narrow and wide vectors, or different stages may involve varying degree of parallelism so that the hardware configuration should be adapted accordingly online to satisfy the varying requirements. As the traditional hardware is absent from capabilities like nested or dynamic parallelism, programmers have to carefully design and tune their programs to avoid hardware resource underutilization.

Fourthly, as the frequency and the number of transistors significantly increase in CPU and accelerators, the heat dissipation and power requirements increase as well. It becomes unacceptable in in modern scientific applications that are usually deployed on large scale clusters equipped with hundreds or even thousands of nodes. Power consumption should be reviewed in modern times as environment-friendly is one of most important issues for sustainable earth.

Fifthly, as accelerators are originally designed for specific purposes, coding for them has always posed a special challenge for software developers due to great differences in hardware architectures across accelerator vendors, or even across generations of accelerations from the same vendor. For example, programming on GPU requires knowledge about image rendering and the GPU hardware specifics. The programming languages such OpenGL or GLSL requires users to abstract the general-purpose applications to fit for GPU rendering model. Besides, all these accelerators have their own Instruction Set Architecture (ISA) that is different from CPU's ISA (like x86). Therefore, communication between CPU and accelerators can be only achieved by dedicatedly designed interfaces and controlled by users. This inevitably imposes additional overhead to users. Optimal performance cannot be guaranteed.

To address the above issues, vendors have revisited the traditional architectural design and released an innovative architecture, where accelerators (like GPU and FPGA) are more tightly coupled with CPU in the same die, eliminating the PCI-e bus by incorporating much faster data paths. Moreover, some new features are introduced to enable software developers with more programming flexibility to deploy applications on such coupled architectures to exploit the great computing power while avoiding the long standing performance bottlenecks.

## 1.2 Existing products

As a promising architectural design trend, hardware vendors have released various coupled architectures to satisfy different requirements.

Since the release of Nvidia G80, GPU has become a popular accelerator to speed up applications with intensive computations and high parallelism on large amount of data. With the increasing adoption of coupled architectures, coupled CPU-GPU architectures demonstrate the advantages over the discrete platforms. In 2011, AMD released its first generation Accelerated Processing Units (formerly known as *Fusion*). In Fusion, the CPU and the GPU are integrated in the same chip, sharing the memory subsystem including main memory and Last Level Cache (LLC). Intel released its early stage products Sandy Bridge (in 2011) and Ivy Bridge (in 2012). In these products, CPU and integrated GPU are collected with ring bus which is collected to the main memory and LLC, so that data sharing between the CPU and the GPU becomes more efficient. Later, in its Iris series products, embedded Dynamic Random-Access Memory (eDRAM) is introduced into system to work as a L4 cache with much larger size (128MB) to share data between the CPU and the GPU. Nvidia also proposed a *Denver Project* with intention to integrate its ARM-based CPU with a broad range of accelerators such as GPU and Digital Signal Processor (DSP).

Here maybe Zeke could help to verify as he is familiar with FPGA. FPGA is an integrated circuit designed to be configured by customers after manufacturing, making it possible to be adapted according to the dynamics to achieve the optimal processing performance on each specific algorithm. Usually it is connected to the CPU via PCI-e bus. In 2016, Intel has launched a processor which integrates Broadwll Xeon with Altera Arria 10 FPGA. Quick Path Interconnect (QPI) is used to connect these hardware components for fast and efficient communication.

Should there be a table? What contents?

## 1.3 Abstract Coupled Architectures

Heterogeneous architectural designs are emerging in the field of computer architecture. Researchers have been proposing different heterogeneous designs in the modern/future processors, which attempt to improve the performance, reduce the energy consumption or both. Among these designs, coupled architectures are becoming increasingly important across a large amount of applications running on platforms ranging from servers to embedded devices.

Though there are some differences in the techniques used by vendors to implement their own coupled architectures, their general design methodologies are similar with each other. We illustrate the abstracted coupled architecture design in Figure 1. As shown in Figure 1, CPU and accelerator are connected

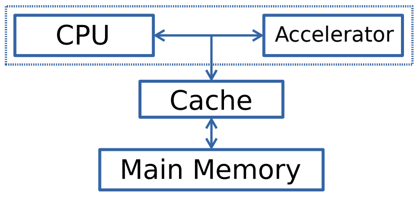

Figure 1: Abstract coupled architectures.

with specialized interconnection techniques (such as QPI, ring bus or unified north bridge). In such way, data transfers are more efficient and flexible than ever before, enabling users put more efforts on the logic design. The memory system including cache and main memory is also shared between CPU and accelerator. With various coherence reserving policies, the shared cache can be used as a working sharing hub between two sides with appropriate algorithmic design.

To ease software developers from error-prone coding on cross platforms, a unified programming langauge, Open Computing Language (OpenCL), originally proposed by Apple Inc. becomes the standard programming language on heterogeneous platforms consisting of CPUs, GPUs, DSPs, FPGAs and other processors or hardware acceleratros. OpenCL programs can be coded once and run on any OpenCL-compatible devices. Existing studies [3, 15] have shown that programs in OpenCL can achieve very close performance to those in platform-specific languages such as CUDA for NVIDIA GPUs and OpenMP for CPUs. For example, Fang et al. [11] demonstrate that the CUDA-based implementations are at most 30% better than OpenCL-based implementations on NVIDIA GPUs. On CPUs, OpenCL even outperforms OpenMP in many applications [15].

All OpenCL-compatible devices are mapped to the same logical architecture as shown in Figure 2, namely compute device. Each compute device consists of a number of Compute Units (CUs). Furthermore, each CU contains multiple processing elements running in the SPMD style. As the CPU is also OpenCL-compatible, systems consisting of CPU and other OpenCL-compatible accelerators are treated as two coupled OpenCL devices. The code piece executed by a specific device is called a kernel. A kernel employs multiple work groups for the execution, and each work group contains a number of work items. A work group is mapped to a CU, and multiple work items are executed concurrently on the CU. The execution of a work group on the target architecture is vendor-specific. For instance, AMD usually executes 64 work items in a wavefront and NVIDIA with 32 work items in a warp. All the work items in the same wavefront run in the Single Instruction Multiple Data (SIMD) manner. On FPGA, each OpenCL kernel is compiled to a custom circuit. The operations within the kernel are implemented using the ALMs and DSPs, forming sub-circuits that are wired together according to the data flow of the algorithm. Load/store units allow access to different memory areas.

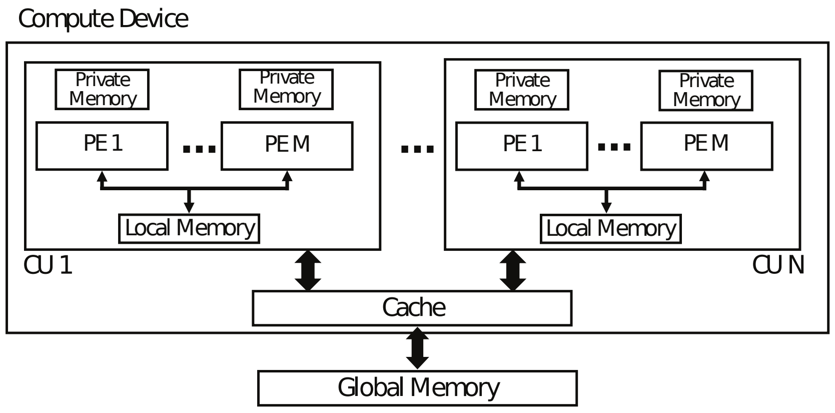

Figure 2: Hardware abstraction in OpenCL.

## 2 Related Work

We review the related work about applications on coupled architectures and in-memory databases on emerging hardware.

### 2.1 Applications on Coupled Architectures

As a novel design, coupled architectures have been widely applied to various fields such as relational databases and big data processing.

#### 2.1.1 Databases

Query co-processing on GPUs is a classic topic that has been exhaustively studied in research community [4, 6, 7, 13, 14, 16]. With the increasing adoption of coupled CPU-GPU architectures, databases on coupled CPU-GPU architectures [8, 18, 9] are also becoming attractive research topics. He et al. [8] proposed a novel fine-grained workload scheduling to optimize hash joins on coupled CPU-GPU architectures. Zhang et al. [18] implemented a fully-fledged query processing system named OmniDB that is able to schedule workload to all available devices in coupled architectures at different levels of granularity such as kernel, operator and query. Furthermore, He et al. [9] further improved their query processing engine by utilizing the shared cache to further reduce memory access latency. All of these query processing systems have demonstrated significant performance improvement over the discrete architectures.

#### 2.1.2 Data Processing

Except the databases on coupled architectures, big data processing can also benefit from this novel architecture design. MapReduce [2] is a popular programming framework for processing large data sets with parallel and distributed algorithms to exploit the great computing power of clusters in a scalable fashion. Though Hong et al. [12] has implemented a MapReduce framework named MapCG on discrete CPU-GPU architectures to accelerate MapReduce applications using both CPU and discrete GPU. However, no consistent speedups over a single device can be obtained as data transfers between the host and the GPU and frequent kernel launches incur significant overhead. Later, Chen et al. [1] proposed a MapReduce framework built on coupled CPU-GPU architectures with flexible workload scheduling algorithms enabled by the shared memory. As the results on 4 applications show, their system achieves 1.21 to 2.1 speed-up over the best performance of the CPU-only and GPU-only versions. The speed-up over a single CPU core execution can even achieve 28.68. Hetherington et al. [11] evaluated a widely used key-value store middleware application, Memcached, on both discrete and coupled CPU-GPU architectures.

Besides, graph processing is another important field that benefits from coupled architectures. Graphs are common data structures in various applications such as social networks, chemistry and web link analysis. The efficiency of graph processing is a must for high performance of the entire system. Existing studies such as Medusa [19] has simplified the programming to solve common graph computation tasks by leveraging the power of GPUs. Lin et al [10] proposed an adaptive algorithm to automatically find the optimal algorithm on the suitable devices of the coupled architecture. 1.6X speedup can be obtained compared with the state-of-the-art algorithms in an energy consumption index, namely TEPS/Watt(Traversed Edges Per Second every Watt). Farooqui et al. [5] conducted a more comprehensive evaluation over a mix of graph applications on coupled CPU-GPU architectures. As the experimental results show, their system Luminar can obtain improvements in both throughput and energy efficiency.

Tsinghua Zhang et al. [17]

#### 2.1.3 Architecture-conscious Design

### 2.2 In-memory Databases on Emerging Hardware

#### 2.2.1 CPU-based Databases

#### 2.2.2 GPU-based Databases

## 3 Tentative Proposals

### 3.1 Pipelined Design

### 3.2 In-cache Design

## 4 Author's Bio

Dr. He Jiong is a researcher in Advanced Digital Sciences Center (ADSC), a joint research center established by the University of Illinois at Urbana-

Champaign and the Agency for Science, Technology and Research (A*STAR), a Singapore government agency. He received his bachelor degree from East China University of Science and Technology (2007  2011) in China. After that, he got the PhD degree from Nanyang Technological University (2011  2016) in Singapore, where his major research interests focus on High Performance Computing (HPC) and database systems. In particular, he interests in applying HPC techniques (like GPGPU) to accelerate the performance of relational query processing and other data-related applications. He has conducted a comprehensive and systematic study on combining HPC and relational database systems by exploiting the power of emerging hardware such as coupled CPU-GPU architectures, which has inspired the database community in building high-performance and energy-efficient data processing systems based on the next generation hardware. In 2013, he was awarded with the VLDB travel grant for his research about hash joins on coupled CPU-GPU architectures. Moreover, his research work has been published in prestigious international proceedings such as VLDB/PVLDB and ACM SIGMOD.

## References

[1] L. Chen, X. Huo, and G. Agrawal. Accelerating mapreduce on a coupled cpu-gpu architecture. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, pages 25:1–25:11, 2012.

[2] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, 2004.

[3] J. Fang, A. L. Varbanescu, and H. Sips. A comprehensive performance comparison of cuda and opencl. In *Proceedings of the 2011 International Conference on Parallel Processing*, ICPP '11, pages 216–225, 2011.

[4] W. Fang, B. He, and Q. Luo. Database compression on graphics processors. *Proc. VLDB Endow.*, pages 670–680, 2010.

[5] N. Farooqui, I. Roy, Y. Chen, V. Talwar, and K. Schwan. Accelerating graph applications on integrated gpu platforms via instrumentation-driven optimizations. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 19–28, 2016.

[6] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational query coprocessing on graphics processors. *ACM Trans. Database Syst.*, pages 21:1–21:39, 2009.

[7] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, and P. Sander. Relational joins on graphics processors. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 511–524, 2008.

[8] J. He, M. Lu, and B. He. Revisiting co-processing for hash joins on the coupled cpu-gpu architecture. *Proc. VLDB Endow.*, pages 889–900, 2013.

[9] J. He, S. Zhang, and B. He. In-cache query co-processing on coupled cpu-gpu architectures. *Proc. VLDB Endow.*, pages 329–340, 2014.

[10] L. Heng, Z. Jidong, and C. Wenguang. Energy-efficient graph traversal on integrated cpu-gpu architecture. SC '15, 2015.

[11] T. H. Hetherington, T. G. Rogers, L. Hsu, M. O'Connor, and T. M. Aamodt. Characterizing and evaluating a key-value store application on heterogeneous cpu-gpu systems. In *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems & Software*, ISPASS '12, pages 88–98, 2012.

[12] C. Hong, D. Chen, W. Chen, W. Zheng, and H. Lin. Mapcg: Writing parallel program portable between cpu and gpu. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, pages 217–226, 2010.

[13] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk. Gpu join processing revisited. In *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, DaMoN '12, pages 55–62, 2012.

[14] 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. *Proc. VLDB Endow.*, pages 1378–1389, 2009.

[15] T. Krishnahari and S.R.Sathe. Comparison of openmp and opencl parallel processing technologies. *IJACSA*, 2012.

[16] H. Pirk, S. Manegold, and M. Kersten. Accelerating foreign-key joins using asymmetric memory channels. In *In VLDB - Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 2011.

[17] F. Zhang, J. Zhai, W. Chen, B. He, and S. Zhang. To co-run, or not to co-run: A performance study on integrated architectures. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*, pages 89–92, 2015.

[18] S. Zhang, J. He, B. He, and M. Lu. Omnidb: Towards portable and efficient query processing on parallel cpu/gpu architectures. *Proc. VLDB Endow.*, pages 1374–1377, 2013.

[19] J. Zhong and B. He. Medusa: Simplified graph processing on gpus. *IEEE Transactions on Parallel and Distributed Systems*, pages 1543–1552, 2014.