Rebooting the Data Access Hierarchy of Computing Systems

Wen-mei W. Hwu
ECE and Coordinated Science Lab
University of Illinois
Urbana, IL 61801
w-hwu@illinois.edu

Izzat El Hajj
ECE and Coordinated Science Lab
University of Illinois
Urbana, IL 61801
elhajj2@illinois.edu

Simon Garcia de Gonzalo
CS and Coordinated Science Lab
University of Illinois
Urbana, IL
grcdgnz2@illinois.edu

Carl Pearson
ECE and Coordinated Science Lab
University of Illinois
Urbana, IL
pearson@illinois.edu

Nam Sung Kim
ECE and Coordinated Science Lab
University of Illinois
Urbana, IL
nskim@illinois.edu

Deming Chen
ECE and Coordinated Science Lab
University of Illinois
Urbana, IL
dchen@illinois.edu

Jinjun Xiong
T.J. Watson Research Center
IBM Corporation
Yorktown Heights, NY
jinjun@us.ibm.com

Zehra Sura
T.J. Watson Research Center
IBM Corporation
Yorktown Heights, NY
zsura@us.ibm.com

Abstract—We have been experiencing two very important movements in computing. On the one hand, a tremendous amount of resource has been invested into innovative applications such as first-principle-based methods, deep learning and cognitive computing. On the other hand, the industry has been taking a technological path where application performance and energy efficiency vary by more than two orders of magnitude depending on their parallelism, heterogeneity, and locality. We envision that a "perfect storm" is coming because of the interaction between these two movements. Many of these new and high-valued applications need to touch a very large amount of data with little data reuse and data movement has become the dominating factor for both power and performance of these applications. It will be critical to match the compute throughput to the data access bandwidth and to locate the compute near data. Much has been and continuously needs to be learned about algorithms, languages, compilers and hardware architecture in this movement. What are the killer applications that may become the new driver for future technology development? How hard is it to program existing systems to address the data movement issues today? How will we program these systems in the future? How will innovations in memory devices present further opportunities and challenges in designing new systems? What is the impact on long-term software engineering cost of applications? In this paper, we present some lessons learned as we design the IBM-Illinois C3SR (Center for Cognitive Computing Systems Research) Erudite system inside this perfect storm.

Keywords—big data, heterogeneous computing, low-complexity algorithms, memory bandwidth, throughput oriented computing

I. Hardware Advances

Although the clock frequencies of processors have not increased in any significant way since 2003, the use of throughput-oriented computing, SIMD execution, and multicore architectures has allowed the computer industry to continue to increase the peak arithmetic computation rate of processors. For example, G80 [1], the first CUDA-enabled GPU from NVIDIA had a peak arithmetic computation rate of 345.6 single-precision GFLOPS. The recent Volta GPU [2] has a peak arithmetic computation rate of 14.03 single-precision TFLOPS. There has been a 41× increase in peak arithmetic computation rate while the clock frequencies of both processors are quite comparable. Such dramatic increase in peak arithmetic computation rate has exerted a tremendous amount of pressure on the memory system, whose data access bandwidth has not increased nearly as much.

The memory bandwidth of G80 was 86.4GB/s, or 21.6 giga single-precision operands per second. This means that for an application to achieve peak arithmetic computation rate of G80, it should perform 16 (345.6/21.6) single-precision floating-point operations for each operand fetched from its GDDR memory system. In Volta, the memory bandwidth is approximately 900 GB/s (or 225 giga single-precision operands per second), thanks to the High Bandwidth Memory (HBM) technology that 2.5D-stacks the DRAM on the processor chip and dramatically increased the number of pins and thus the available bandwidth. There has been a 10.4× increase in memory bandwidth. It is clear that the increase in memory bandwidth has not kept up with the increase in arithmetic computation rate between these generations of GPUs. As a result, in Volta, for an application to achieve peak arithmetic computation, it should perform 62.3 (14,030/225) single-precision floating-point operations for each operand fetched from its HBM memory system. That is, the achieved computation rate of applications on Volta heavily depends on their achieved level of data reuse.

II. Application Trends

The advances in computer hardware, as exemplified by the increase of peak arithmetic computation rate and memory bandwidth from G80 to Volta, allow application developers in many domains to use brute-force, first-principle-based methods which employ physically sound and mathematically rigorous formulations without making any fundamental approximation [3]. These methods avoid the errors and inaccuracies introduced by previous approximate methods and produce higher quality results.

The continued growth of computing power has also motivated developers to solve bigger problems. For example, in weather forecast, the applications are using increasingly finer grid-point spacing in their models to improve the accuracy of the prediction, which translates into much larger problems for the PDE solvers. For another example, a text mining application may apply Latent Semantic Analysis (LSA) [4] on a larger number of documents to enlarge the scope of search, which results in larger matrix decomposition problems.

Traditionally, applications have used direct solvers on dense matrices. These direct solvers exhibit data reuse and access locality that can be exploited with on-chip memories to reduce off-chip memory bandwidth consumption. Data reuse is key to achieving a high percentage of the peak arithmetic computation rate in modern processors. Unfortunately, direct solvers also exhibit high computational complexity of $O(N^2)$ or even $O(N^3)$. They do not scale well to very large problems. One way to think about it is that for an $O(N^2)$ algorithm, a 100× increase in computation rate only allows a 10× increase in problem size.

Therefore, practical use of rigorous methods that requires solving large problems is a result of not only the computer power we have today, but also fast (low-complexity) algorithms with O(N) or O(N×log(N)) computational complexities that have been developed in the past 20 years. These algorithms, such as multigrid methods, fast multipole methods, and hierarchical matrix methods provide solutions to many science and engineering problems which were previously thought to be intractable. Without fast algorithms, even the world's largest supercomputers would not be able to solve even some of the modest-sized problems that can now be solved with low-complexity algorithms on a powerful workstation.

Another important challenge is that as problem sizes grow, the models tend to become increasingly sparse. For example, in LSA, the document-term matrix becomes sparser as the number of documents increases. This behavior is because each term appears only in a subset of the documents. As the number of documents increases, the contents of these documents tend to diversify. Thus, the document-term matrix tends to become increasingly sparse. Thus, these matrices are best stored as sparse matrices.

It is well-known that direct solvers are not suitable for inversing or decomposing sparse matrices because of the large number of fill-in elements, the new non-zero elements generated in the process of matrix inversion or decomposition. Iterative solvers are commonly used for sparse matrices. In many applications, iterative solvers often employ low-complexity algorithms to perform the forward calculations in each iteration.

Unfortunately, low-complexity algorithms achieve their low computational complexity by processing each piece of data only once or a very small number of times. Therefore, these low-complexity algorithms intrinsically exhibit little data reuse. When these low-complexity algorithms are used in iterative solvers to solve very large problems, the memory access pattern becomes multiple sweeps through large data structures with very little reuse within each sweep.

As a result, the execution speed of applications that solve large problems using low-complexity algorithms is typically limited by off-chip memory bandwidth (i.e., memory bound). For example, from G80 to Volta, the achieved execution rate of these applications tends to only increase by 10× (improvement in memory bandwidth) rather than 41× (improvement in peak arithmetic computation rate).

In terms of achieved arithmetic computation rate for iterative solvers using low-complexity algorithms, one can expect it to be close to 225 GFLOP (the maximal rate of fetching operands form memory since there is little or no data reuse), rather than the 14 TFLOPS. As a result, these applications typically achieve only 1-2% of the peak arithmetic computation rate of modern processors.

III. Very Large Problems and Heterogeneity

Modern GPUs come with an ample amount of memory. For example, Volta comes with 24GB of HBM. While 24 GB is a large amount of memory, it is not sufficient for solving very large problems. Many applications solving large problems require hundreds of gigabytes or even terabytes of memory. The typical way of executing such applications on GPUs in a heterogeneous computing system is to access data from the host memory or even the disk storage.

Unfortunately, the access path between the host memory and the GPU operates at about 80GB/s for NVLINK and 8 GB/s for PCIe 3. That is, the data access speeds through NVLINK and PCIe 3 are approximately 10% and 1% of the Volta off-chip memory bandwidth. With NVLINK, the achieved arithmetic computation rate for Volta will likely be less than 20 GFLOPS. With PCIe, the achieved arithmetic computation rate for Volta will likely be less than 2 GFLOPS. That is, when an application on the GPU accesses its data from the host memory, its achieved arithmetic computation rate could be 0.1-0.2% of the peak for NVLINK and 0.01-0.02% of the peak for PCIe3.

If the host system does not have enough memory to hold the data set, the application will have to access the data set from disk storage. This will likely cause another significant drop in achieved computation rate due to the overhead of deserialization when reading data from files.

IV. Application Workflows

Another important trend in application development is to organize large applications into microservices that form a workflow. Each microservice performs a well-defined analysis and/or transformation on the data and deliver the data to the downstream services. This software architecture allows application developers to leverage standard software components and reduce the well-known risks of large, monolithic software. However, the workflow model can incur significant overhead in modern computing systems. The data being passed from one service to the next is transferred as files. Large data objects need to be serialized into files by the sender and deserialized into memory objects by the receiver. The serialization and deserialization overhead can sometimes exceed the actual processing time [5]. Also, the deserialization overhead can introduce unacceptable latency in interactive systems. Such overhead can be potentially eliminated by passing the data sets as persistent memory objects when the sender and the receiver reside in the same computer.

V. Rebooting the Data Access Hierarchy

In this section, we describe our proposal to redesign the data access hierarchy of future computing systems to drastically improve their achieved computation rate of applications while solving very large problems.

A. *Eliminating Files*

We expect that the memory and storage of future computing systems will converge with the use of non-volatile memory technology. Non-volatile memory does not require refresh and allows a much higher level of integrated capacity than the DRAM technology. We expect that the capacity of the non-volatile memory to be comparable to the solid-state disk (SSD) of the current generation of computers. Therefore, we propose to store the large data sets as persistent objects rather than traditional files, which was pioneered by the designers of IBM AS/400 [6].

Maintaining data sets as persistent objects eliminates the serialization and deserialization overhead for workflow software architectures. Furthermore, maintaining inference models as persistent memory objects can eliminate much of the latency involved in interactive cognitive applications. More importantly, accessing large persistent objects from memory allows for a higher level of parallelism than traditional file systems. However, there is a significant challenge in mapping the addresses (pointer values) of the objects into the virtual address spaces of multiple processes. The Erudite persistent memory objects are based on the SpaceJMP work [7] that allows each process to freely switch between virtual address spaces that serve as containers of large data sets. A new directory mechanism is being developed to enable fast discovery and mapping of persistent objects.

B. *Near Memory Acceleration*

An important innovation in the Erudite system design is Near Memory Accelerators (NMAs) [8]. The idea is to place computing devices near the memory chips so that the compute can access data at a higher aggregate rate than what the traditional shared memory channels can support for processors. These NMA units will be designed to issue a large number of requests to the memory chips in order to tolerate the long access latency. By placing the NMA units close to the memory chips, they can have local channels that support much higher aggregate bandwidth than a traditional memory channel shared by many memory devices.

These NMA units will be exposed to the host processor in two modes. In the first mode, a special device driver makes the existing software stack recognize NMA units as (heterogeneous) processors connected over traditional communication channels/protocols such as Ethernet and PCIe [9]. This mode allows existing applications which were developed to utilize a traditional communication channel to run without any modification at the application level. We refer to NMA units in this mode as Memory Channel Network (MCN) processors. MCN processors can support distributed computing applications in popular frameworks such as SPARK without any change to the application code as the host processor sees MCN processors as traditional network-connected slave nodes. The second mode is a processor that communicates with the host processor through memory channel protocols. NMA units in the second mode are called NMA devices. NMA devices can collaborate with the host processors and GPUs in a heterogeneous system just like any other accelerators in a heterogeneous system architecture. The first prototype of an MCN processor is under development based on the IBM ConTutto memory controller board [9].

MCN processors and NMA devices will likely consist of programmable throughput-oriented processors as well as application-specific hardware [9]. The programming and high-level synthesis for these processors and devices is supported by Tangram [10], a programming system that enhances performance portability of applications by synthesizing device-specific code.

C. *Collaborative Heterogeneous Computing*

Erudite is designed as a heterogeneous computing system where CPUs, GPUs, FPGAs, and NMA devices can engage in fine-grained or coarse-grained collaborative execution. The fine-grained collaboration support is based on Chai [11], where a library of fine-grained queues and synchronization primitives is under development for devices to deliver their execution results to collaborators and release dependent activities waiting for these results.

Chai currently supports CPUs, GPUs, and FPGAs connected through PCIe. We are extending Chai to support NMA devices connected through memory channels. The main use case is for NMA devices to perform computation on large data sets with little or no data reuse. Using the Chai support, these NMA devices can deliver their results to CPUs, GPUs, or FPGAs to perform computation that better matches the capability of these devices.

D. *Cognitive Computing Applications.*

The design of Erudite is initially driven by two cognitive computing applications. The first application is DISCVR, a natural-language document understanding application for advanced queries and automated reviews/summaries. This system stores a massive number of documents, performs large-scale sparse matrix decomposition operations, solves large constrained problems, and performs matching and traversal operations on large graphs. The second application is CELA, an interactive system that serves as a teaching assistant to help guide students through experiential learning processes such as science experiments, cooking lessons, hands-on training, and language labs. This system stores a large data base of curriculum, knowledge base, and project descriptions. It performs speech processing, video processing, natural language dialog, synthesis of new learning projects or labs, and human activity comprehension. These two applications contain a wealth of computation methods and solvers that are commonly used in other cognitive applications. As we progress, we will consider adding new driving applications.

Many cognitive computing workloads, such as speech and video processing mentioned above, contain deep neural networks (DNN) as the underlining machine learning engine. Because DNNs are very computationally intensive, part of our effort is to design systems to accelerate important DNN workloads. Recently, we targeted the Long-term Recurrent Convolutional Network (LRCN) [12]. LRCN is among the most complex tools available today aiming to achieve cognitive intelligence for video content analysis. In this work, we used an FPGA as the customizable platform. Several ideas are explored in our design, such as reuse of common patterns of computation, on-chip double buffering to hide memory access latency, computation resource allocation according to the data and computation demands of the CNN and RNN layers, and bringing as much data as possible to on-chip memories for fast data-accessing speed. Results show that our design is 3.1x faster and 17.5x more energy efficient than a GPU implementation, where the FPGA and the GPU have similar transistor counts. The ideas and concepts used in this initial work are being adopted into the Erudite system design.

VI. Conclusion

In this paper, we present our rationale for the Erudite system design. We argue that data access bandwidth limitations in current processors result in extremely low utilization of GPUs when solving large problems with low-complexity algorithms. We further examine the application trend that will make solving large problems with low-complexity algorithms even more important in the future. We then give a high-level description of the Erudite system that is being developed jointly between Illinois and IBM to drastically increase the achieved computation rate of cognitive applications that solve large problems using low-complexity algorithms. We believe that the combination of persistent objects, NMA, and collaborative heterogeneous computing will be required to give future computing systems a new platform that can support continued growth of performance and energy efficiency for these high-valued, demanding applications.

Acknowledgment

The authors would like to thank members of the IBM-Illinois Center for Cognitive Computing Systems Research (C3SR) for their contributions to the formation of the Erudite system. This work is in part sponsored by the IBM-Illinois C3SR, the Starnet Center for Future Architecture Research (C-FAR), and E2CDA-NRI, a funded center of NRI, a Semiconductor Research Corporation (SRC) program sponsored by NERC and NIST under Grant No. NERC 2016-NE-2697-A and by the National Science Foundation under Grant No. ECCS 16-40196.

References

[1] Wikipedia contributors. "GeForce 8 series." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 29 Aug. 2017. Web.

[2] NVIDIA Corporation. NVIDIA Tesla V100 GPU Architecture Whitepaper. 2017.

[3] Hwu, Wen-mei, et al. "Thoughts on massively-parallel heterogeneous computing for solving large problems." Computing and Electromagnetics International Workshop (CEM), 2017. IEEE, 2017.

[4] Susan T. Dumais, "Latent Semantic Analysis". Annual Review of Information Science and Technology. 38: 188–230. doi:10.1002/aris.1440380105, 2005.

[5] El Hajj, Izzat, et al. 2017. "SAVI Objects: Sharing and Virtuality Incorporated." PACM Progr. Lang. 1, OOPSLA, Article 45 (October 2017), 24 pages.

[6] Wikipedia contributors. "AS/400 Object." The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 27 June 2016. Web.

[7] El Hajj, Izzat, et al. "SpaceJMP: programming with multiple virtual address spaces." ACM SIGOPS Operating Systems Review 50.2 (2016): 353-368.

[8] H. Asghari-Moghaddam, et al., "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," IEEE/ACM International Symposium on Microarchitecture (MICRO), 2016, pp. 1-13.

[9] N.S. Kim, et al., "Heterogeneous Computing Meets Near-Memory Acceleration and High-Level Synthesis in the Post-Moore Era," IEEE Micro, 37(4), pp. 10—18, July/August 2017.

[10] Chang, Li-Wen, et al. "Efficient kernel synthesis for performance portable programming." Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016.

[11] Gómez-Luna, Juan, et al. "Chai: collaborative heterogeneous applications for integrated-architectures." Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on. IEEE, 2017.

[12] X. Zhang, et al., "High-Performance Video Content Recognition with Long-term Recurrent Convolutional Network for FPGA", FPL, 2017.