Parallel Processing with Big Data

Behrooz Parhami
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA

Synonyms

Big-data supercomputing; Computational needs of big data

Definition

Discrepancy between the explosive growth rate in data volumes and the improvement trends in processing and memory access speeds necessitates that parallel processing be applied to the handling of extremely large data sets.

Overview

Both data volumes and processing speeds have been on exponentially rising trajectories since the onset of the digital age (Denning and Lewis 2016), but the former has risen at a much higher rate than the latter. It follows that parallel processing is needed to bridge the gap. In addition to providing a higher processing capability to deal with the requirements of large data sets, parallel processing has the potential of easing the "von Neumann bottleneck" (Markgraf 2007), sometimes referred to as "the memory wall" because of its tendency to hinder the smooth progress of a computation, when operands cannot be supplied to the processor at the required rate (McKee 2004; Wulf and McKee 1995). Parallel processing algorithms and architectures (Parhami 1999) have been studied since the 1950s as a way of improving computer system performance and, more recently, as a way of continuing the exponential rise in performance while keeping the power consumption in check (Gautschi 2017; Gepner and Kowalik 2006; Koomey et al. 2011).

Trends in Parallel Processing

Interest in parallel processing began in the 1960s with the design of ILLIAC IV, generally recognized as the world's first supercomputer (Hord 2013). The 64-processor machine, later built and operated by Burroughs Corporation, had a single-instruction-stream, multiple-data-stream architecture, SIMD for short (Flynn and Rudd 1996), which uses a single instruction execution unit, with each instruction applied to multiple data items simultaneously. The other main class of parallel architectures is known as MIMD, in which there are multiple instruction streams in addition to multiple data streams. The latter architectural category has a great deal of variation in terms of how memory is implemented (global

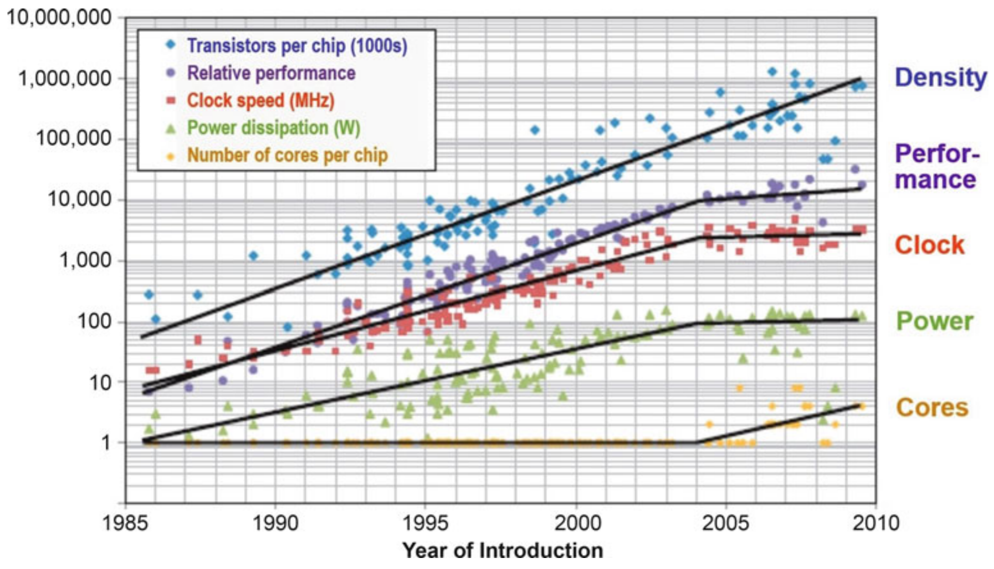
© Springer International Publishing AG, part of Springer Nature 2018
S. Sakr, A. Zomaya (eds.), *Encyclopedia of Big Data Technologies*,
https://doi.org/10.1007/978-3-319-63962-8_165-1

or distributed) and how the multiple processors communicate and synchronize with each other (shared-variable or message-passing).

Modern supercomputers, including most entries in the Top500 Supercomputers List, which is updated twice a year (Top-500 2017), tend to be MIMD DMMP, meaning that they use distributed memory, whereby processors are endowed with chunks of the memory space, and communicate via message-passing. Each processor has direct (and thus fast) access to its local memory and indirect (and thus slower) access to nonlocal or remote chunks of memory. Top-of-the-line supercomputers are often used to perform numerically intensive computations involving floating-point numbers. Thus, their performance is measured in floating-point operations per second, or FLOPS. As of late 2017, the top supercomputer in the world had a peak performance of 125 PFLOPS (Peta-FLOPS) and a sustained performance at 75% of the peak. We are now on the verge of achieving a peak performance of 1 EFLOPS (Exa-FLOPS or 10^{18} FLOPS).

The topmost trend line in Fig. 1 shows the exponential rise in the number of transistors that can be placed on a single microchip. This exponential rise in density is known as Moore's Law, after Intel co-founder Gordon E. Moore, who first observed the trend (Brock and Moore 2006; Mack 2011; Schaller 1997). Up to the early 2000s, the rising chip density was accompanied by exponential improvement in performance, owing to the correspondingly higher clock frequencies. Then, clock frequency and performance trend lines began to flatten out, mostly because the attendant growth in power dissipation led to difficulties in cooling the superdense circuits. Thus, focus on performance enhancement shifted to using multiple independent processors on the same chip, giving rise to multicore processors (Gepner and Kowalik 2006). Using c cores, each with $1/c$, the performance is more energy-efficient, because power consumption is a superlinear function of the single-core performance.

Big data necessitates a reassessment of how we gauge supercomputer performance. While FLOPS rating and IOPS, its integer counterpart, are still important, input/output and storage bandwidth assume a larger role in determining performance. A supercomputer doing scientific calculations may receive a handful of input parameters and a set of equations that define a scientific or engineering model and then compute for days or weeks, before spewing the answers.



Parallel Processing with Big Data, Fig. 1 Trends in transistor density, processor performance, clock frequency, power dissipation, and number of cores on a chip. (NRC 2011)

Many big-data applications require a steady stream of new data to be input, stored, processed, and output, thus possibly straining the memory and I/O bandwidths, which tend to be more limited than computation rate.

Parallel Processing Models

Parallel processing architectures vary greatly in their organizations and user interfaces. Over the years, many abstract models of parallel processing have been introduced in an effort to accurately represent processing, storage, and communication resources, along with their interactions, allowing developers of parallel applications to visualize and take advantage of the available trade-offs, without being bogged down in machine-specific details. At the coarse level, we can distinguish data-parallel and control-parallel approaches to parallel processing (Parhami 1999).

Data parallelism entails partitioning a large data set among multiple processing nodes, with each one operating on an assigned chunk of data, before participating in the process of combining the partial results. It is often the case in big-data applications that the same set of operations must be performed on each data chunk, making SIMD processing the most efficient alternative. In practice, however, synchronization of a large number of processing nodes introduces overheads and inefficiencies that cut into the speed gains. So, it might be beneficial to direct the processing nodes to execute a single program on multiple data chunks asynchronously, with sparse coordination, giving rise to the SPMD (Darema 2001) parallel processing model.

Google's MapReduce (Dean and Ghemawat 2008, 2010) is the most prominent cloud-based system for SPMD parallel processing. In the map stage, data is distributed within independent tasks on different processing nodes. Each map-stage task produces a set of key-value pairs as its output. These outputs are then fed to reduce-stage tasks, where they are aggregated into a smaller set of key-value pairs that constitute the final output. Hadoop is a widely used open-source implementation of MapReduce on Apache servers (Bu et al. 2010). It consists of Hadoop's version of MapReduce, the Hadoop distributed file system (Shafer et al. 2010; Shvachko et al. 2010), that allows a user to store huge data files across multiple nodes while keeping the image of a centrally located file (stored in one piece), and Hadoop YARN (Vavilapalli et al. 2013), which manages computing resources and allows tasks to be scheduled on them.

The fully open-source Apache Spark (Zaharia et al. 2016) is in many ways similar to MapReduce, with the main differences that it uses a unified engine to replace various other needed subsystems, thus making the programming effort much more manageable and less error-prone, and a data-sharing abstraction, "resilient distributed data sets" or RDDs, which make it significantly more efficient for certain workloads.

In control-parallel schemes, which are more broadly applicable than data parallelism, multiple nodes operate independently on subproblems, synchronizing with each other by sending informational and synchronization messages. The said independence allows heterogeneous and application-specific resources to be employed, without cross-interference or slower resources hindering the progress of faster ones. More on this later. An attractive submodel of control parallelism, that aims to reduce communication and I/O overheads, is bulk synchronous parallel, or BSP, processing (Valiant 1990). BSP computations consist of communication- and synchronization-free supersteps that are executed to conclusion independently, before any communication or synchronization occurs.

Google's Pregel system (Malewicz et al. 2010) is a practical implementation of BSP for executing iterative graph algorithms, while holding an entire large graph in memory, spread across many nodes, so as to avoid disk access latency. Other examples of the control-parallel paradigm are seen in event processing and stream processing systems commonly used in social media and other notification-driven applications (Abadi et al. 2003; Condie et al. 2010; Eugster et al. 2003; Rixner 2001).

Beyond the two broad kinds of parallel processing, reflected in data-parallel and control-parallel schemes, there are various other kinds of parallelism that can be used as competing or complementary approaches. These include instruction-level parallelism (Rau and Fisher 1993), subword parallelism (Lee 1997), dataflow parallelism (Sakai et al. 1989), and simultaneous-multithreaded parallelism (Eggers et al. 1997), the latter offering the advantage of lower overhead as well as greater resource sharing in running interdependent tasks in parallel.

Heterogeneity and Hardware Acceleration

Heterogeneous parallel processing entails using multiple processing resources that have diverse designs and performance parameters. Heterogeneity complicates the problems of task assignment and coordination, but it also increases flexibility in matching computational requirements of tasks to hardware and software capabilities. In a heterogeneous system, users can have access to the capabilities of a supercomputer, say, even though their use of such a powerful machine is too limited to justify purchase or even a long-term usage contract.

Within a heterogeneous parallel computing system, some nodes may be specialized for highly efficient execution of certain computations through the use of application-specific hardware/-software and augmentation with accelerators. Examples include use of graphic processing units or GPUs as computation engines (Nvidia 2016; Owens et al. 2008; Singer 2013), units optimized for information retrieval and data mining (Sklyarov et al. 2015), and a variety of other specialized resources made accessible through the cloud (Caulfield et al. 2016).

In addition to fixed acceleration resources, reconfigurable devices, that can be programmed to act as different kinds of acceleration mechanisms, may be deployed. Modern FPGAs (Kuon et al. 2008) offer immense computational resources that can be tailored dynamically to help remove processing bottlenecks. Application-specific circuits, whether custom-designed or implemented on FPGAs, provide additional options for acceleration. For example, sorting networks (Mueller et al. 2012; Parhami 1999) can be used in a wide array of contexts, which include facilitating classification or speeding up subsequent search operations.

Interconnection Networks

Modern parallel computers use commodity processors, often multicore, multithreaded, or GPUs, as computing resources. This allows rapid advances in processor speed and energy efficiency to become immediately available to parallel systems. It follows that the distinction between various parallel processing systems is, to a great extent, determined by the interconnection networks used to allow data exchange between computing nodes. The network connecting processing resources can also be commodity, such as Ethernet. However, it often pays to design a custom network, or at least use a custom configuration of commodity switches, that matches the communication requirements.

Over the years, numerous interconnection network architectures have been proposed and used (Dally and Towles 2004; Duato et al. 2003; Parhami 1999) for different system scales. When computing nodes are on the same electronic chip, they are connected by an on-chip network (Benini and De Micheli 2002), whose design is often more restricted in view of limitations on area and power. Interconnecting processing nodes within a large mainframe or supercomputer system presents challenges in wiring and packaging, with many theoretically superior topologies becoming impractical because they cannot be implemented within the physical limits imposed by partitioning, packaging, signal delays on long wires, and the like (Dally and Towles 2004). Interconnecting servers within a data center presents fewer limitations in terms of topology, but given the scale, energy efficiency considerations, as well as reliability and serviceability factors, become dominant.

Mapping, Scheduling, and Virtualization

A fundamental problem in parallel processing is the mapping of computations to hardware resources. Ideally, this mapping should be automatic and transparent, so as to relieve users from having to deal with hardware details and configuration changes. However, user-guided mapping can result in better running times and more efficient resource utilization, in view of targeted optimizations in fitting parts of the tasks to capabilities of the available processing resources.

An important kind of mapping, whose roots go back to before computers appeared on the scene, is scheduling. For parallel processing, scheduling refers to the assignment of computational subtasks to processing nodes while honoring various constraints, in such a way that an objective function is optimized (Sinnen 2007). The simplest objective function is task completion time, but a variety of more detailed objectives, such as meeting deadlines and minimizing energy consumption, may also enter into the picture.

Virtualization (Nanda and Chiueh 2005) allows system resources to be used effortlessly among different users while isolating and protecting them from one another to ensure privacy and security. The techniques used represent extensions of, and improvements upon, time-sharing schemes of the 1960s (Wilkes 1972) that fell out of favor when compact and inexpensive hardware became available in the form of minicomputers and, eventually, microcomputers. Virtual-machine monitors (Rosenblum and Garfinkel 2005) isolate users from hardware details, resource fluctuations, and configuration changes, thus allowing greater focus on application correctness and high-level trade-offs. Reliability is also improved, both because software bugs are isolated within virtual machines, preventing their spread, and because any detected hardware malfunction can be circumvented by rescheduling affected tasks on other operational virtual machines.

Future Directions

While parallel processing technologies have matured over more than five decades, requirements of big-data applications are already creating new challenges, which will pose greater difficulties with the continued exponential growth in data volumes. This explains the proliferation of proposals for hardware architectures, software aids, and virtualization schemes to ease the key bottlenecks.

Studies on expected direction of computer architecture research over the coming decades (Ceze et al. 2016; Stanford University 2012) point to substantial growth in the use of parallel processing for performance, scalability, energy economy, and reliability reasons. Changes in parallel architectures and attendant virtualization techniques will both influence and be driven by the evolution of cloud computing systems (Agrawal et al. 2011). As data volumes grow and the scope of hardware resources for big-data processing expands, energy efficiency, which is already a major design and cost consideration, will grow in importance (Koomey et al. 2011).

Cross-References

- ▶ Big Data and Exascale Computing
- ▶ Computer Architecture for Big Data
- ▶ Energy Implications of Big Data
- ▶ GPU-Based Hardware Platforms

References

Abadi DJ, Carney D, Cetintemel U, Cherniack M, Convey C, Lee S, Stonebraker M, Tatbul N, Zdonik S (2003) Aurora: a new model and architecture for data stream management. Int J Very Large Data Bases 12(2): 120–139

Agrawal D, Das S, El Abbadi A (2011) Big data and cloud computing: current state and future opportunities. In: Proceedings of 14th international conference on extending database technology, Uppsala, pp 530–533

Benini L, De Micheli G (2002) Networks on chips: a new SoC paradigm. IEEE Comput 35(1):70–78

Brock DC, Moore GE (eds) (2006) Understanding Moore's law: four decades of innovation. Chemical Heritage Foundation, Philadelphia

Bu Y, Howe B, Balazinska M, Ernst MD (2010) HaLoop: efficient iterative data processing on large clusters. Proc VLDB Endowment 3(1–2):285–296

Caulfield AM et al (2016) A cloud-scale acceleration architecture. In: Proceedings of 49th IEEE/ACM international symposium microarchitecture, Orlando, pp 1–13

Ceze L, Hill MD, Wenisch TE (2016) Arch2030: a vision of computer architecture research over the next 15 years, Computing Community Consortium. Online document. http://cra.org/ccc/wp-content/uploads/sites/2/2016/ 12/15447-CCC-ARCH-2030-report-v3-1-1.pdf

Condie T, Conway N, Alvaro P, Hellerstein JM, Elmeleegy K, Sears R (2010) MapReduce online. Proc USENIX Symp Networked Syst Des Implement 10(4):20

Dally WJ, Towles BP (2004) Principles and practices of interconnection networks. Elsevier, Amsterdam

Darema F (2001) The SPMD model: past, present and future. In: Proceedings of European parallel virtual machine/message passing interface users' group meeting, Springer

Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113

Dean J, Ghemawat S (2010) MapReduce: a flexible data processing tool. Commun ACM 53(1):72–77

Denning PJ, Lewis TG (2016) Exponential laws of computing growth. Commun ACM 60(1):54–65

Duato J, Yalamanchili S, Ni LM (2003) Interconnection networks: an engineering approach. Morgan Kaufmann, San Francisco

Eggers SJ, Emer JS, Levy HM, Lo JL, Stamm RL, Tullsen DM (1997) Simultaneous multithreading: a platform for next-generation processors. IEEE Micro 17(5): 12–19

Eugster PT, Felber PA, Guerraoui R, Kermarrec A-M (2003) The many faces of publish/subscribe. ACM Comput Surv 35(2):114–131

Flynn MJ, Rudd KW (1996) Parallel architectures. ACM Comput Surv 28(1):67–70

Gautschi M (2017) Design of energy-efficient processing elements for near-threshold parallel computing. Doctoral thesis, ETH Zurich

Gepner P, Kowalik MF (2006) Multi-core processors: new way to achieve high system performance. In: Proceedings of IEEE international symposium on parallel computing in electrical engineering, Bialystak, pp 9–13

Hord RM (2013) The Illiac IV: the first supercomputer. Springer, Berlin

Koomey JG, Berard S, Sanchez M, Wong H (2011) Implications of historical trends in the electrical efficiency of computing. IEEE Ann Hist Comput 33(3):46–54

Kuon I, Tessier R, Rose J (2008) FPGA architecture: survey and challenges. Found Trends Electron Des Autom 2(2):135–253

Lee RB (1997) Multimedia extensions for general-purpose processors. In: Proceedings of IEEE workshop signal processing systems, design and implementation, Leicester, pp 9–23

Mack CA (2011) Fifty years of Moore's law. IEEE Trans Semicond Manuf 24(2):202–207

Malewicz G, Austern MH, Bik AJC, Dehnert JC, Horn I, Leiser N, Czajkowski G (2010) Pregel: a system for large-scale graph processing. In: Proceedings of ACM SIGMOD international conference on management of data, Indianapolis, pp 135–146

Markgraf JD (2007) The von Neumann bottleneck. Online source that is no longer accessible (will find a replacement for this reference during revisions)

McKee SA (2004) Reflections on the memory wall. In: Proceedings of the conference on computing frontiers, Ischia, pp 162–167

Mueller R, Teubner J, Alonso G (2012) Sorting networks on FPGAs. Int J Very Large Data Bases 21(1):1–23

Nanda S, Chiueh TC (2005) A survey on virtualization technologies, technical report TR179, Department of Computer Science, SUNY at Stony Brook

NRC (2011) The future of computing performance: game over or next level? Report of the US National Research Council, National Academies Press

Nvidia (2016) Nvidia Tesla P100: infinite compute power for the modern data center – technical overview. http://images.nvidia.com/content/ tesla/pdf/nvidia-teslap100-techoverview.pdf. Accessed 14 Dec 2017

Owens JD et al (2008) GPU computing. Proc IEEE 96(5):879–899

Parhami B (1999) Chapter 7: Sorting networks. In: Introduction to parallel processing: algorithms and architectures. Plenum Press, New York, pp 129–147

Rau BR, Fisher JA (1993) Instruction-level parallel processing: history, overview, and perspective. J Supercomput 7(1–2):9–50

Rixner S (2001) Stream processor architecture. Kluwer, Boston

Rosenblum M, Garfinkel T (2005) Virtual machine monitors: current technology and future trends. IEEE Comput 38(5):39–47

Sakai S, Hiraki K, Kodama Y, Yuba T (1989) An architecture of a dataflow single chip processor. ACM SIGARCH Comput Archit News 17(3):46–53

Schaller RR (1997) Moore's law: past, present and future. IEEE Spectr 34(6):52–59

Shafer J, Rixner S, Cox AL (2010) The Hadoop distributed filesystem: balancing portability and performance. In: Proceedings of IEEE international symposium on performance analysis of systems & software, White Plains, pp 122–133

Shvachko K, Kuang H, Radia S, Chansler R (2010) The Hadoop distributed file system. In: Proceedings of 26th symposium on mass storage systems and technologies, Incline Village, pp 1–10

Singer G (2013) The history of the modern graphics processor, TechSpot on-line article. http://www.techspot.com/article/650-history-of-the-gpu/. Accessed 14 Dec 2017

Sinnen O (2007) Task scheduling for parallel systems. Wiley, Hoboken

Sklyarov V et al (2015) Hardware accelerators for information retrieval and data mining. In: Proceedings of IEEE conference on information and communication technology research, Bali, pp 202–205

Stanford University (2012) 21st century computer architecture: a community white paper. http://csl.stanford.edu/~christos/publications/2012.21 stcenturyarchitecture.whitepaper.pdf

Top-500 Organization (2017) November 2017 list of the world's top 500 supercomputers. http://www.top500.org/lists/2017/11/

Valiant LG (1990) A bridging model for parallel computation. Commun ACM 33(8):103–111

Vavilapalli VK et al (2013) Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of fourth symposium on cloud computing, Santa Clara, p 5

Wilkes MV (1972) Time-sharing computer systems. Elsevier, New York

Wulf W, McKee S (1995) Hitting the wall: implications of the obvious. ACM Comput Archit News 23(1):20–24

Zaharia M et al (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65