

High performance concepts

by Jack Hinton

FILE	1568_JACK_HINTON_HIGH_PERFORMANCE_CONCEPTS_94244_653319 249.DOCX (22.12K)		
TIME SUBMITTED	22-JAN-2018 11:29AM (UTC+0000)	WORD COUNT	2914
SUBMISSION ID	82199067	CHARACTER COUNT	15768

High performance concepts

By Jack Hinton – p147676

Lecturer: Dr. Tim Bashford

Module: Distributed and Cluster Computing

Historical context of parallelism	3
Parallel programming paradigms.	4
Threading	5
Definition	5
Description	5
Benefits	6
Drawbacks	6
Cluster processing	7
Definition	7
Description	7
Benefits	7
Drawbacks	8
GPGPU	8
Definition	8
Description	8
Benefits	Error! Bookmark not defined.
Drawbacks	9
Some peculiarities of parallel computing	9
Race conditions	10
Deadlocks	10
Amdahl's law	10

Historical context of parallelism

'Parallelism' or 'parallel computing' is a term used to describe the practice of running or creating processes that contain operations able to be simultaneously performed. Although the practice of parallelism has become increasingly popular in recent years, the concept originated in 1842 in L. F. Menabrea's "Sketch of the analytical engine Invented by Charles Babbage" [1].

Menabrea describes a process by which the operation upon and duplication of a set of entered numbers may be pipelined as to occur simultaneously. This process prevents the user from having to enter the same set of numbers more than once for use in many operations, and reduces both the chance of human error and the total runtime per input.

Whilst this was a necessary optimisation at the time, the advent of digital computing temporarily offset this requirement due to the fact that the speed with which data could be entered and operations could be performed was massively increased. Though the original electronic digital computers such as ENIAC used a form of parallelism [2], subsequent computers most often scheduled operations using a more serial approach, the exception to this being with regards to input or output. [3]

Although the first commercially available parallel computer was launched in 1952 [4], the need for widespread parallel computing was not universally acknowledged until much later, when it was realised that solitary processing units were likely to soon reach their maximum speed in terms of clock rate and floating-point operations per second (FLOPS)[3]. In recognition of this it was determined that the most efficient way to increase computational speed was now to add additional processing units, an initiative now known as multiprocessing.

The advent of multiprocessing had a great impact on the design of both hardware and software. As the speed of a CPU vastly outstripped that of any other component, CPU-specific memory had to be increased in order to reduce slowdown caused by storage read-times [5]. To allow the cores to communicate without unnecessary latency, 'bridges' had to be created between the cores that ran at a speed comparative to the cores themselves. In order to facilitate the collaboration of the cores on single tasks, the availability of fast memory accessible by the multiple cores became more important. This created a need for software which was able to encompass the asynchronous nature of access to these memory banks, known collectively as 'caches', and to be able to efficiently split the list of tasks so that they could be assigned to multiple cores.

Cache

'Cache' is a term commonly used to refer to fast access-rate memory that is reserved solely for use by the CPU in order to speed up operations performed. A cache may be used as a sort of buffer, where sizeable chunks of relevant data are stored in the hope that they may be useful (a 'cache hit'), or to contain values that are generated by the CPU whilst performing an operation. One example of the former might be reading the next N values of a list when the first item is requested as it is likely that the rest will be needed subsequently. One example of the latter might be to contain a loop counter during a mean average operation. Caches are organised into 'levels' of speed, with the highest (level 1) being physically connected to the CPU, often to an individual core. In modern CPUs the level 2 caches are normally connected to each core's level 1 cache [6], whereas the level 3 cache is separate and shared by all cores. Cache architecture is designed in this way to allow a tiered approach to reading from it - if data is required by a core, the highest level cache is read. If the data is not found, the lower level caches are read in succession until finally the main storage is consulted.

Bridges

A 'bridge' is a term commonly used to describe a chip facilitating the connection between the CPU, its attendant ram and the motherboard. In many architectures there are two bridges, referred to as the 'Northbridge' and the 'Southbridge' [7]. The Northbridge runs at a clock speed which is only slightly less than the CPU cores themselves, and is used to allow rapid communication between the cores and the faster caches. The Southbridge runs significantly slower than the Northbridge, and is used to convey data from and to the motherboard. Due to this it is often considered to be the 'I/O relay' for the CPU. It is worth noting however, that this architecture has recently been modified by Intel so as to include the Northbridge within the die of the CPU [8], now known as 'sandy bridge'. This has occurred in order to reduce the quantity of separate components placed on a motherboard, and thus decrease manufacturing cost. [9]

Parallel programming paradigms.

Threading

Definition

'Threading' is a term used to refer to the practice of separating a program into multiple distinct control flows or 'threads', which are largely independent of one another [10]. These threads may then run concurrently and thus can greatly increase a process' overall speed of execution. Threads have access to a public memory bank and thus can share data between each other [11], although care must be taken to ensure that this shared memory is not adversely affected by asynchronous access.

Description

Most modern operating systems make extensive use of threading in order to streamline the user experience [12]. A simple process such as Microsoft Notepad may contain only one thread, whereas a more complex process such as Google Chrome may contain many threads performing different functions. A thread that is managed by the operating system is known as a 'kernel thread' and is typically generated on boot. Threads managed by user-controlled programs are known as 'user threads', and are mapped to a free kernel thread when they are executed. [13]

The process of creating and optimising threads so that they may run in tandem is often referred to as 'multithreading'. Separate but related to this is 'interleaved multithreading', where multiple virtual processors are simulated on one core and are scheduled so as to minimise the impact of latency caused by memory reads. This differs from standard multithreading as the emphasis in this scenario is now on creating blocks of read/write operations across all interleaved threads, rather than on asynchronous processing. [14]

This approach can be further broken down into 'fine-grained' multithreading (where threads are switched between in a round-robin fashion), 'coarse-grained' multithreading (where threads are switched if a particularly slow read occurs), 'time-slice' multithreading (where threads are switched between after a set time has elapsed) and 'switch-on-event' multithreading (where threads are switched between if the current thread has to wait for input).



Benefits

Allows simultaneous completion of tasks without the use of specialist hardware. Provides a conceptually unchallenging approach to parallelism, thus allowing the programmer to create more powerful solutions.

Drawbacks

All threads within a process are affected by the state of global variables and settings within this process. [15]

If a thread performs an illegal operation and ends, the process to which the thread belongs will also end.

Cluster processing

Definition

'Cluster processing' is a term used to refer to the practice of linking multiple computers together to form a larger 'super-computer'. In this scenario, each networked device can be regarded as analogous to a 'core' in a single computer.

Description

When designing a computer cluster, the physical layout and description of the component machines must be carefully considered with respect to the tasks the completed system will be expected to perform. Responsibilities that require a disparate and unconnected series of events (such as running a web-server) may not necessitate homogeneity of component devices, whereas functionality with a high level of inter-process communication (such as complex modelling procedures) may demand a greater level of coupling and thus component machines of similar specification.

Computer clusters may be constructed to perform a variety of tasks, but the emphases with which they are constructed fall into two main categories; load-balancing and high-availability.

A high-availability or 'failover' cluster is constructed to ensure that the service provided is uninterrupted regardless of circumstance. It achieves this by creating simple virtual machines to serve requests rather than serving them all from the main operating system. If one of these machines fails, a duplicate may be quickly made and resume the set task.

A load-balancing cluster attempts to ensure all component machines within the cluster have an equal share of the workload in order to maximise the efficiency of execution.

Parallelism in these systems is commonly accomplished using the Message Passing Interface, or MPI. MPI is built around the principle of using data packets sent between processes to both synchronise and allow them to communicate [16]. This allows for efficiency on both a local and global scale in homogenous and heterogeneous clusters alike, as local scheduling can be delegated to the component machines whilst allowing supervision by an overarching management protocol.

Benefits

One of the benefits of MPI is its portability. As it relies on a simple concept, it can be implemented efficiently on a great range of hardware. MPI(2) contains support for remote-memory operations and analogues for UNIX-type file operations, thus allowing it to be implemented for different operating systems [17]. Furthermore, MPI allows for easy manipulation of data regardless of locality, and is able to compensate for differing hardware speeds on various networked computers. Additionally, MPI is relatively efficient as it allows

programmers to treat machines as individual units rather than sections of the whole machine, one may optimise for that unit. This division allows for machine-specific peculiarities to be addressed.

Drawbacks

MPI has limited support for shared-memory operations, and thus using MPI to implement a large-scale application with shared memory may require more complexity than other approaches.

GPGPU

Definition

General Programming on Graphics Processor Units (GPGPU) is the practice of running programs using a computer's GPU rather than its CPU. As graphics processors are purpose built to facilitate the simultaneous processing of a great number of matrix operations, this can dramatically increase the performance of programs that operate in a compatible manner.

Description

GPGPU is now a widely-used approach to parallelism, due to the fact that GPUs commonly possess a great number of homogeneous cores and thus from a conceptual standpoint, are easy to write parallel programs for.

GPGPU was first attempted when DirectX 8 became available [18] as there were now programmable vertex and pixel shading routines within the graphics card. Initially, the only way to perform GPGPU was through the graphics API, so algorithms to be performed had to be presented to the graphics card as if they were required for rendering.

At this point, the functionality presented by GPGPU was minimal for a number of reasons. Firstly, the locations within graphics memory were allocated and managed by the GPU, meaning that algorithms requiring random locations within memory could not be run. Furthermore, there was little in the way of a standardised approach to floating-point arithmetic within a graphics processing unit, and thus scientific calculations could not be guaranteed to run on any particular machine. Finally, If the program crashed or failed, there was little to no way that the programmer could debug the erroneous code.

These problems were then addressed in 2006, when NVidia released their first graphics processor built using the CUDA architecture. This architecture was designed in part to facilitate the use of the graphics processor for general purpose programming by allowing the reprogramming of many of the pipelines within the card [15]. In addition, the onboard ALUs were built to comply with the IEEE recommendations for floating point arithmetic, and thus were now

reliably usable in scientific calculations. Finally, NVidia worked to allow developers to use C in order to interface with the graphics card, rather than having to use a shader language through DirectX or OpenGL.

GPGPU actually uses a system that is somewhat of a fusion of MPI and threading, as it uses 'blocks' or processes that may communicate with each other using messages, and also allows division of a 'block' into many threads that communicate with each other using shared memory. [19]

Benefits

GPGPU allows for an extreme increase in operational speed due to the architecture of the utilised GPU.

Drawbacks

As graphics processors are constructed solely to perform matrix calculations, any program intended for implementation using GPGPU must present all operations as a series of matrix calculations. This may increase complexity in situations where the data types used are not natively suited to this expression.

Some peculiarities of parallel computing

Race conditions

A race condition is formed when a program's output varies depending on the order in which instructions given are carried out. One example of this was found in the Therac-25 Medical Accelerator, which would emit a lethal dose of radiation if the user first selected 'X-ray' mode, then rapidly selected 'treatment' mode [20]. The initial selection of X-ray mode would position the beam direction magnets so that they would not interfere with the beam, and the second selection would set the beam to be used in high-power mode rather than low power. These two conditions together resulted in an undirected emission of radiation at a lethal amplitude.

Deadlocks

A deadlock is a situation in which two or more threads are waiting for the others to complete their respective tasks [21]. As all threads are waiting for the other to signal them, none will progress and thus the system will stall. One example of this occurred in the North American version of the game 'Bubble Bobble Revolution'. This deadlock was caused by a physical defect that prevented an enemy from spawning. The game would then be unplayable as the player would not be able to progress until the errant enemy was defeated

Amdahl's law

Amdahl's law states that parallelism may only ever increase the speed with which a program completes to the limit imposed by the combined completion times of its serial components.[22]

This means that when a parallel program is being created, care must be taken to minimise the quantity of serial tasks that are performed in order to gain the greatest increase in speed. In this case, the descriptor 'serial task' also applies to tasks bounded by synchronisation operations.

Bibliography

- [1] L. Menabrea, C. Babbage, A. Lovelace and A. L., Sketch of the Analytical Engine invented by Charles Babbage. R. & J.E. Taylor: London, 1843.
- [2] C. Langton, Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems. Cambridge, Mass. [u.a.]: MIT Press, 1997, p. 9.
- [3] "Concurrent processing: A new direction in scientific computing", in National Computer Conference, Chicago, Illinois, 1985, pp. 157-165.
- [4] C. Bashe, W. Buchholz, G. Hawkins, J. Ingram and N. Rochester, "The Architecture of IBM's Early Computers", IBM Journal of Research and Development, vol. 25, no. 5, pp. 363-376, 1981.
- [5] A. Danowitz, K. Kelley, J. Mao, J. Stevenson and M. Horowitz, "CPU DB: Recording Microprocessor History", Queue, vol. 10, no. 4, p. 10, 2012.
- [6] S. Cho and L. Jin, "Managing Distributed, Shared L2 Caches through OS-Level Page Allocation", 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), 2006.
- [7] P. Conway and B. Hughes, "The AMD Opteron Northbridge Architecture", IEEE Micro, vol. 27, no. 2, pp. 10-21, 2007.
- [8] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann and D. Rajwan, "Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge", IEEE Micro, vol. 32, no. 2, pp. 20-27, 2012.
- [9] W. Wolf, "The future of multiprocessor systems-on-chips", Proceedings of the 41st annual conference on Design automation - DAC '04, pp. 681-685, 2004.
- [10] A. Butterfield and G. Ekembe Ngondi, A dictionary of computer science, 7th ed. Oxford: Oxford University Press, 2016, p. 557
- [11] Massachusetts Institute of Technology, "Traffic control in a multiplexed computer system.", Massachusetts Institute of Technology. Dept. of Electrical Engineering, Massachusetts, 1966.
- [12] "Thread Management", Developer.apple.com, 2018. [Online]. Available: <https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Multithreading/CreatingThreads/CreatingThreads.html>. [Accessed: 18- Jan- 2018].

- [13] "Beyond Multiprocessing : Multithreading the SunOS Kernel", in USENIX Summer 1992 Technical Conference, Texas, 1992, pp. 11-18.
- [14] J. Laudon, A. Gupta and M. Horowitz, "Interleaving", ACM SIGPLAN Notices, vol. 29, no. 11, pp. 308-318, 1994.
- [15] W. Gropp, E. Lusk and A. Skjellum, Using MPI. Cambridge, MA: MIT Press, 1999.
- [16] T. Sterling, "An Introduction to PC Clusters for High Performance Computing", Cluster Computing White Paper, p. 7, 2000.
- [17] R. Keller, Recent advances in the message passing interface. Berlin: Springer, 2010, pp. 161-171.
- [18] J. Sanders, E. Kandrot and J. Dongarra, CUDA by example. Upper Saddle River [etc.]: Addison-Wesley/Pearson Education, 2015.
- [19] ZIB, "Multi-threaded Kernel Offloading to GPGPU Using Hyper-Q on Kepler Architecture", Zuse Institute Berlin, Berlin, 2014.
- [20] N. Leveson and C. Turner, "An investigation of the Therac-25 accidents", Computer, vol. 26, no. 7, pp. 18-41, 1993.
- [21] C. Ellis, "On the probability of deadlock in computer systems", ACM SIGOPS Operating Systems Review, vol. 7, no. 4, pp. 88-95, 1973.
- [22] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", IEEE Solid-State Circuits Newsletter, vol. 12, no. 3, pp. 19-20, 2007.

High performance concepts

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

82/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5



Comment 1

Chrome is an interesting example because, while it obviously is threaded, it's nowhere near as threaded as it could be! It tends to create a lot of separate processes to circumvent the potential instability of any given web page. So if, for example, some bad JavaScript gets into an infinite loop, the process can be killed without impacting other pages. Also allows tabs to swap between windows.

This is a complete aside - you're certainly not wrong (Chrome uses a few threads per process), but Chrome itself is an interesting and quite esoteric parallel implementation!



Comment 2

There should be an inline citation for this paragraph.

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

GRADING FORM: SMU GENERIC_GJ

JACK HINTON**82****RESEARCH-INFORMED LITERATURE**

Excellent use of literature - well done, Jack. An exceptional number of sources, nearly all of high academic credibility. To improve this further, I would suggest you try to improve the density of your inline citations in places - the GPGPU and clustering sections especially are a tad sparse. Also try to break the habit of "say one thing, cite one source" - it's far better to read multiple sources on the same topic, compose your argument, then cite them all [1][2][3][4]. Regardless, excellent work.

8.5**KNOWLEDGE AND UNDERSTANDING**

Robust knowledge and understanding demonstrated - good, logical and structured information presented coherently, with appropriate depth and breadth of reading - really well done.

23**ANALYSIS**



Good synthesis of material, however conclusions (including mini section conclusions) are a tad conspicuous in their absence. These would actually help you to tie your report together better, so sections flow together more easily into a more cohesive narrative.

17

PRACTICAL APPLICATION AND DEP.



Good inclusion of discussion of relevant technology, while maintaining a strong academic tone. This could maybe be slightly better-emphasised given the large weighting of marked assigned to it, but this is a very minor point - consider using the mark scheme as almost a weighting guide for formatting the report.

25

SKILLS FOR PROFESSIONAL PRAC.



You write extremely well - I really enjoyed reading this report. You might find a figure or two would help you better-emphasise your points.

8.5