How Task-Based and Concurrent Algorithms are currently used to great affect within industry

Anthony Sturdy

Task-Based Software Engineering

Staffordshire University

Concurrent Programming is a method of developing an application that allows multiple tasks to be executed in parallel. True parallelism cannot be performed on a single core processor, the CPU will switch between tasks, but they won’t be processed concurrently unless they are being executed on a multi-core CPU (Clifton, 2008). The advantage of parallel processing is that more of the CPU cores are utilised at once meaning tasks are processed faster.

Although parallel processing can increase an application’s speed, it can also be equal or slower than sequential processing if done wrong. Sometimes the overhead of starting a task on a new thread can cause the task to execute slower than sequential processing. Another common problem with parallel programming is when race conditions occur, this can happen when two separate threads operate on the same object at the same time (Paul, 2017) which can lead to the application crashing (Rouse, 2019).

Most modern processors have multiple cores and multiple threads. According to the Steam March 2020 Hardware and Software Survey (Steam, 2020), only 0.41% of Steam’s user base use single core processors on Windows, and 1.32% on Linux. This means the utilisation of parallel processing is important to software developers as it could greatly increase the speed of applications over most devices. Parallel programming is most important when quick processing time is vital, for example, when working with big data. If the purpose of an application is to filter through or process 100,000,000 pieces of data, it could take a long time sequentially, whereas if it were processed concurrently it may be sped up by a considerable amount.

Another example where parallel programming is important is within video games. Modern video games often aim to render 60 frames per second, meaning everything in that frame needs to be processed in under 0.016 seconds. That’s all the input, AI, audio, networking, graphics, etc. in under 16 milliseconds. This would be a lot of work for just a CPU, but with a dedicated graphics card, graphics can be rendered a lot quicker. Graphics cards are designed to process in parallel (Hruska, 2018), which is why it’s able to render 2,073,600 pixels for a 1080p render in such a short amount of time. Pieces of code, called Shaders, are sent to the graphics card, as well as data to be processed. When pixels are being rasterised (by what is known as a Fragment Shader), the code can’t access the data from neighbouring pixels due to it being ran parallel. This means GPU’s are very fast, and are incredibly useful for many different applications, not just rendering.

Plenty of industries use parallel programming (Gossett, 2019): Astronomers need to simulate objects in space due to how slowly events happen in space. Computers use parallel processing to quickly and accurately simulate these events; Banks use GPUs for things such as fraud detection to credit scoring. Crypto mining also caused a shortage of graphics cards in 2018 causing GPU prices; the medical industry uses GPUs for high computation and high bandwidth tasks which has led to better speed and accuracy for many aspects of the medical industry; these are just a few examples but there are many more instances where parallel programming can be useful.

Microsoft offer multiple tools for parallel programming, such as F#, the Task Parallel Library and LINQ/PLINQ (Clifton, 2008). F# itself isn’t a language for parallel programming, but it does support asynchronous workflows which “can be used to write applications that have UIs that remain responsive to users as the application performs other work” (Microsoft, 2016). The Task Parallel Library brings functionality to .NET languages which simplify a parallel programming workflow. Instead of the developer having to create functionality to start new threads and tasks, TPL provides most of the functionality already. Developers can easily Parallelise a for loop by using Parallel.For instead of just a regular for loop, for example. LINQ (Language-integrated Query) or the Parallel version, PLINQ, can be used to query data similarly to SQL. Parameters are set on the query, and any objects which meet the parameters will be returned in a new collection. These tools offered by Microsoft make .NET a great platform for parallelism, but other languages can also utilise concurrent processing. For example, Cpp TaskFlow can be used for C++, Numba can be used for Python, and there are many other libraries offering similar functionality to simplifiy parallel programming for other languages.

Cluster computers are a group of computers linked together in order to work as a single computer (Hussein, 2009) and are often highly parallelised to gain the optimal performance out of the multiple CPU’s. Beowulf is an approach to building a supercomputer as a cluster of other computers connected via a LAN (Local Area Network) to run programs written for parallel processing. “The Beowulf idea is said to enable the average university computer science department or small research company to build its own small supercomputer that can operate in the gigaflop (billions of operations per second) range” (Rouse, 2006). The Beowulf architecture is found in approximately 88% of the worlds largest parallel computing systems (Bate, 2019). Beowulf enables anyone to create a cluster computer and is commonly created with Raspberry Pi’s as a learning experience. Although Raspberry Pi’s are not very fast alone, they are easily accessible and cheap, which makes them great for producing a DIY cluster computer. For example, Josh Kiepert built a cluster PC with 33 Pi’s and custom chassis to hold them, custom PCB’s to distribute power, RGB LED’s and fans to keep the cluster cool (Szczys, 2013). That is not all necessary though, as Simon Cox built a similar cluster computer using 64 Raspberry Pi’s configured for parallel processing, held together by a chassis made of Lego. Although that is a lot of Raspberry Pi’s all working together, it is still not close to being in the top 500 supercomputers, as an estimated 1.4 million Raspberry Pi’s would be needed to outrank the 500th fastest supercomputer (Benchoff, 2012).

In conclusion, parallel programming is used in many different industries and is extremely useful for utilising a CPU’s full speed. It is useful in applications for general use as most modern CPU’s have multiple cores, and also useful for computationally intensive tasks such as simulations and processing large amounts of data. Sequential processing is still useful for situations where the overhead of starting new tasks is greater than the efficiency of parallelism, and a large amount of applications will not need to utilise as much of the CPU as possible to achieve the fastest processing times as it can. Parallel programming is extremely useful in a lot of cases, but not always necessary.

Bate, A. (2019). *Beowulf Clusters, node visualisation and more with Pi VizuWall*. [online] Raspberry Pi. Available at: https://www.raspberrypi.org/blog/beowulf-clusters-node-visualisation-pi-vizuwall/ [Accessed 16 Apr. 2020].

Benchoff, B. (2012). *64 Rasberry Pis Turned Into A Supercomputer*. [online] Hackaday. Available at: https://hackaday.com/2012/09/12/64-rasberry-pis-turned-into-a-supercomputer/ [Accessed 16 Apr. 2020].

Clifton, M. (2008). *Concurrent Programming - A Primer*. [online] www.codeproject.com. Available at: https://www.codeproject.com/Articles/22649/Concurrent-Programming-A-Primer [Accessed 14 Apr. 2020].

Gossett, S. (2019). *9 Parallel Processing Examples & Applications*. [online] builtin.com. Available at: https://builtin.com/hardware/parallel-processing-example [Accessed 15 Apr. 2020].

Hruska, J. (2018). *How Graphics Cards Work*. [online] www.extremetech.com. Available at: https://www.extremetech.com/gaming/269335-how-graphics-cards-work.

Hussein, L. (2009). *Parallel & Cluster Computing*. [online] wmich.edu. Available at: https://cs.wmich.edu/elise/courses/cs626/s09/hussein/Parallel\_and\_Cluster\_Computing.pdf [Accessed 16 Apr. 2020].

Microsoft (2016). *Asynchronous Workflows - F#*. [online] docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/asynchronous-workflows [Accessed 15 Apr. 2020].

Paul, J. (2017). *What is Race Condition in multithreading – 2 Examples in Java*. [online] Javarevisited.blogspot.com. Available at: https://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html [Accessed 14 Apr. 2020].

Rouse, M. (2006). *What is Beowulf?* [online] SearchDataCenter. Available at: https://searchdatacenter.techtarget.com/definition/Beowulf [Accessed 16 Apr. 2020].

Rouse, M. (2019). *What is race condition? - Definition from WhatIs.com*. [online] SearchStorage. Available at: https://searchstorage.techtarget.com/definition/race-condition [Accessed 14 Apr. 2020].

Steam (2020). Steam Hardware & Software Survey: March 2020. [online] Available at: https://store.steampowered.com/hwsurvey/cpus/ [Accessed 14 Apr. 2020].

Szczys, M. (2013). *33 Node Beowulf Cluster Built With Raspberry Pi*. [online] Hackaday. Available at: https://hackaday.com/2013/05/21/33-node-beowulf-cluster-built-with-raspberry-pi/ [Accessed 16 Apr. 2020].