COVENTRY UNIVERSITY

6004CEM PARALLEL DISTRUBUTED PROGRAMMING

ID: 7982202

Due Date: 04/23/2021

Contents

COURSE PARALLELISM vs FINE GRAIN PARALLELISM:

In programming there is a process known as parallel computing. Parallel computing allows developers to create programs with work together at the same time, this allows for more computations to be done simultaneously. Parallel computing granularity in terms of tasks, refers to the measurement of the amount of work in context with computation that is performed within the task.

According to Benedict, Lee, David (2013), Coarse grain parallelism is highly arithmetically intense. Coarse grain parallelism works by dividing a program into many large threads/tasks. The tasks are executed on multiple processors simultaneously. As a result of this division, communications between the different tasks is limited as it becomes computationally expensive. The limited communication can be inefficient because one processor might do significantly more work while another processor is not being utilized, thus making it only suitable for message passing architecture (D Grossman, 2012). This can increase the time is takes to complete all the tasks. Coarse grain parallelism can make the process more complex because it divides the tasks in a way that fine grain parallelism does not, which is that it splits the tasks into a few large groups. This means that the processors have an easier time understanding the tasks because they are more together. Coarse grain parallelism makes it difficult to isolate a section of information because of the way coarse grain computes, while this is also true for fine grain parallelism it is significantly easier when dealing with large amounts of information.

According to Benedict, Lee, David (2013) fine grain parallelism has low arithmetic intensity. Fine grain parallelism works by dividing a program into a series of very many small threads. Much like coarse grain parallelism, fine grain assigns tasks to many different processors. However, fine grain differs as it evenly divides the work among each individual processor. This means that unlike coarse grain parallelism, fine grain allows each processor to complete the computation roughly at the same time. Before a computation can be made fine grain must first check the intensity and it must be less intense than coarse grain parallelism as in fine grain parallelism the independent unit has less work as in fine grained parallelism needs less power to operate, whereas coarse grain parallelism needs more power due to the tasks executing independently in larger chunks. Fine grain unlike coarse grain allows the user to balance the load effectively when there are many tasks being computed. Fine grain parallelism offers fast communication, so it is suitable architectures that support fast communication. Since programmers find it difficult to deal with programs that work with parallelism, the compiler must take responsibility for the detection of any fine grain parallelism. (M Chu, 2007)

LIMIT OF PARALLEL PROCESSING:

According to Amdahl's law the speeding up of potential program is defined with the help of fraction of code P which can be parallelized.

Speedup = 1/ 1-P

Just in case none of the code is parallelized then P=0 and speed up = 1

If all the codes are parallelized, then P=1 and speed up is equal to infinity

Just in case 50% of the code is parallelized the maximum value of speed up = 2 which means that the code will run twice and fast.

If number of processors are introduced to perform the fraction of work which is parallel, then the relationship can be modeled as;

Speedup = 1/ P/N+S

Whereas P refers to parallel fraction, N refers to the number of processors and S refers to the serial fraction. There is limitation in terms of scalability of parallelism. A cost optimal parallel program solves the problem with a cost that is proportional to the execution time of the sequential algorithm which is known fastest present on a single processor. (MV da Silva, 2008) The cost of complexity is measured by considering time of a programmer. It is believed by many programmers that performance of a computer program can be increased by performing multiple tasks simultaneously. It is true that parallel programming has multiple advantages and utilization. Parallelism can be performed as a mechanism through which multiple programs can be performed simultaneously. To find out how parallelism can improve productivity of a computer program we can utilize Amdahl's law. Let us assume that 10 processors are running parallel and simultaneously. Consider that parallel part is composed of 60% and sequential part is composed of 40%, a speed of 2.17 can be gain with the utilization of Amshal's law:

SPEED UP(N)= 1/ (1-P) +P/N

1-P = serial part of the job

p/n= parallel part is divided by n workers\

if parallel part is composed of 80% then the value of speed up will be 3.57

if parallel part is composed of 90% then the value of speed up will be 5.26

If parallel part is composed of 99% then the value of speed up will be 9.17.

As it can be seen above even utilization of 99% of parallelism and 10 processors failed to achieve speed up which is 10 times. In the reality achieving 99% of parallelism is very difficult and utilized very rarely and it increases the cost. (A Medina, 2003)

DISTRIBUTION:

Climate models are an extension of weather forecasting. Weather models are meant to make predictions for specific areas and within short time spans. Climate borders can help when analyzing longer time spans. Distributing programming and distributed algorithms refer to a computer network in which multiple individuals are distributed physically within some geographical areas. Multiple hardware and software-based architectures are utilized in distributed computing. At lower levels, it is important to connect numerous CPUs with any network regardless of their physical location (Ad Kshemkalyani, 2011). When it comes to higher levels, the CPU is required to work with some kind of communications system. For example, a client server, three tier, n-tier, or peer to peer are the multiple basic architectures in which distributed programming falls. The diverse nature of an application may ask for the utilization of a communication network.

There are multiple scenarios in which the utilization of a single processor would be possible, but due to the increased efficiency of multiple processors, the utilization of a distributed system is more efficient. Utilizing a cluster of multiple low-end processors instead of using a single high-end processor would not only help in achieving higher performance, but it would also offer cost efficiency (EA Basha, 2008). Even though comprehensive climate models have become more complex, multiple other models have helped by providing solutions to address multiple problems. According to Duffy (2003), the accuracy of simulations cannot simply be improved by increasing the spatial resolution. Climate models rely on parameterization of physical, biological and chemical processes to be able to indicate the effects of sub-grid processes which are unresolved.

According to Bennartz (2011), the assumptions which are made regarding the parameterization are dependent on the scale, even though scale aware development of parameterization has only just been pursued in recent times. An increase in model resolution may lead to degradation of the simulation fidelity. A climate system is composed of a multiple range of processes which are complex thus involving scales which are spatially and temporally composed of orders of different magnitude. Sea ice models, most of the time, consists of processes which are dynamic and thermodynamic. They offer calculations which are improved. The efficiency of a climate model can be increased with hind-casting tests, in which future climate is simulated and the results obtained are valid. The integral component of climate model tends to be the dynamic core that is meant to solve the governing equations of the components involved in the system numerically (H Attiya, 2004).

Increasing the resolution of the model comes at the cost of a considerably higher computational cost. Without achieving concomitant increase in the vertical resolution and the horizontal resolution cannot be achieved in order to avoid the distortion of the results. As the complexity of the model increases, so does the computational cost. There is a need of striking a balance between resolution and complexity. To achieve higher resolution among the computational constrains, multiple other approaches such as variable resolution, stretched grids etc. have been developed to be able to achieve refinement locally for geographical areas. The conservation equation for energy, momentum and water vapor which govern the atmospheric state can be solved numerically and simultaneously by atmospheric models which are global. According to Alex and Andrew (2018), if the data size for the algorithms is increased, the learning errors can be decreased and at times it could prove itself to be more effective than other complex methods. There are numerous systems that can perform tasks in an environment which is distributive. According to Galakatos (2017), node machine learning algorithm can help in improving the performance, which can also help in increasing accuracy along with scaling the data sizes of input. According to J Verbraeken (2019), increased parallelization and bandwidth of input and output can help improve the accuracy of the system which is distributed. Message passing interface tend to be a framework which is low in level, but it is designed to achieve high performance while performing the computation which is distributed. Also, message passing interface offers many primitives which includes sending and receiving which allows the implementation of a wide range of applications. (Kalogirou, 2003)

REFERENCES:

Kalogirou, S. A. (2003). Artificial intelligence for the modeling and control of combustion processes: a review. *Progress in energy and combustion science*, *29*(6), 515-566.

Basha, E. A., Ravela, S., & Rus, D. (2008, November). Model-based monitoring for early warning flood detection. In *Proceedings of the 6th ACM conference on Embedded network sensor systems* (pp. 295-308).

Walthall–Andrew, C. S. D. A., & Wueste–Benjamin, T. E. Preliminary Report on the 2018 Season of the American Excavations at Morgantina: Contrada Agnese Project (CAP).

Attiya, H., & Welch, J. (2004). *Distributed computing: fundamentals, simulations, and advanced topics* (Vol. 19). John Wiley & Sons.

Kshemkalyani, A. D., & Singhal, M. (2011). *Distributed computing: principles, algorithms, and systems*. Cambridge University Press.

da Silva, M. V., & Antão, A. N. (2008). Upper bound limit analysis with a parallel mixed finite element formulation. *International Journal of Solids and Structures*, *45*(22-23), 5788-5804.

Medina, A., Ramos-Paz, A., & Fuerte-Esquivel, C. R. (2003). Periodic steady state solution of electric systems with nonlinear components using parallel processing. *IEEE Transactions on Power Systems*, *18*(2), 963-965.

Grossman, D., & Anderson, R. E. (2012, February). Introducing parallelism and concurrency in the data structures course. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 505-510).

Americans, M. (2013). Chinese Americans, Turkish Germans: Parallels in Two Racial Systems. *Multiple Identities: Migrants, Ethnicity, and Membership*, 290.

Chu, M., Ravindran, R., & Mahlke, S. (2007, December). Data access partitioning for fine-grain parallelism on multicore architectures. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)* (pp. 369-380). IEEE.

Orr, M. S., Beckmann, B. M., Reinhardt, S. K., & Wood, D. A. (2014, June). Fine-grain task aggregation and coordination on GPUs. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)* (pp. 181-192). IEEE.