International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

MULTI CORE PROCESSOR

Shreehari H S[1], Mahesh reddy H L[2]

[1]Assistant Professor, Dept. of ECE, SJCIT, Chikkaballapur, India
[2]Student, Dept. of ECE, SJCIT, Chikkaballapur, India
[1]shreehari.harthi@gmail.com, [2]maheshmahindra4@gmail.com

ABSTRACT

The world we live in has been completely transformed by microprocessors, and ongoing attempts are being made to produce not only faster chips but also smarter ones. The performance of microprocessor cores has been significantly enhanced by a number of approaches, including hyper threading (Intel's HT), instruction level parallelism, and data level parallelism. There are various multicore architectures, each with a distinct level of performance, and because there are so many different architectures, it is vital to compare them all to ensure that the performance matches the desired specifications. The correct methods for choosing multicore CPU performance strategies, illustrations of metrics used in multicore CPU performance studies, elements affecting multicore CPU performance, and benchmarks that focus on multicore CPU performance characteristics.

Keywords *- central processing unit (CPU)*

1. INTRODUCTION

The only method to increase the speed of high-end processors at the moment is to add support for more threads, either by increasing the number of cores or by using multithreading on cores to hide long-latency processes. The previous clock rate gains cannot be maintained for a variety of reasons. Only the most obvious and compelling argument for higher clock rates needs to be made; equally significant is the fact that wire delays rather than transistor switching will be the primary problem for each clock cycle. Multi-core CPUs are much more diversified and have a much larger design area than single-threaded processors. In this chapter, we outline the key scaling concerns, explore some recent examples, and outline the architectural principles of multi-core chip designs. These concepts can now be combined onto a single chip multiprocessor, which is more commonly known as a multi-core processor, thanks to advancements in semiconductor technology. However, when the trade-offs are incorporated onto the same silicon die, they change drastically, as they usually do. Additionally, recent advancements in semiconductor technology have made it possible to have more control and flexibility through isolated islands of voltage and frequency, giving system software the ability to regulate performance and modify power consumption to suit the needs of the moment.

2. METHODOLOGY

The design of a multi-core processor allows for communication between all of the available cores, and it divides and distributes all processing responsibilities effectively. Once all processing activities have been completed, the processed data from each core is sent back to the computer's main board (Motherboard) via a single common gateway. In terms of overall performance, this approach is superior to a single-core CPU.

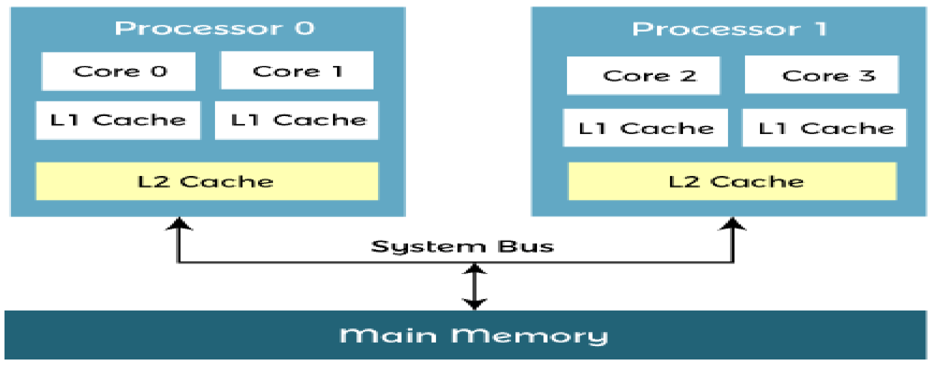

Fig: Core processor

A multi-core CPU can naturally perform more tasks than a single-core processor. In an integrated circuit, the distance between the cores enables higher clock rates. As a result, the signals are persistent and do not need to travel a great distance to reach their destination. The rates are much faster than when using a separate processor. On the other side, a multi-core CPU uses less power when multitasking. We'll just use the portion of the CPU that produces heat. Eventually, the power consumption is reduced, which causes the battery to be used less. On the other side, some operating systems require more resources than others. Multiple resources, both internal and external, are shared by a multi-core CPU. These resources include primary memory, system buses, and networks. As a result, there is a greater possibility that a program operating on the same core will be interrupted. In this kind of interference, isolation can take place both spatially and temporally.

3. MULTIPROCESSOR INTERCONNECT

As core capabilities and speed continued to advance more quickly than memory, the end result was that cores were idle for a sizable portion of the time as they awaited the completion of high latency memory operations. As a result of this discovery, hardware multi-threading was developed, enabling a core to handle several thread contexts in hardware and enable quick switching between hardware threads whenever one or more of the threads became stopped due to operations with a high latency. Numerous implementations of this idea have been offered over time, but the majority of systems in use today employ simultaneous multi-threading, a fine-grained multithreading technique used in the context of out-of-order cores. Instructions from several hardware threads are dispatched in each cycle, or in the case of super-scalar processors, multiple instructions can be dispatched per core cycle, allowing for the concurrent execution of instructions from various threads. The latency of determining where out-of-order execution can be carried out is greatly decreased as instructions from separate threads use distinct registers, resulting in a higher density of effectively executed instructions per cycle.

4. BLOCK DIAGRAM

A multi-core processor is a computer processor that has two or more cores, which are independent processing units that each read and execute program instructions on a single integrated circuit. The instructions are standard CPU commands like add, move data, and branch, but because a single processor can execute commands on several cores simultaneously, applications that support multithreading or other parallel computing techniques run faster overall. The cores are often combined by the manufacturers into numerous integrated circuit dies in a single chip package or onto a single integrated circuit die known as a chip multiprocessor, or CMP. Almost all personal computers today employ multi-core microprocessors. In a single physical package, a multi core CPU implements multiprocessing. Cores in a multi-core device may be coupled tightly or loosely by the designers. For instance, cores may implement message forwarding or shared-memory inter-core communication techniques, and they may or may not share caches. Network topologies like bus, ring, two-dimensional mesh, and crossbar are frequently used to connect cores. Only identical systems are homogeneous multi-core.

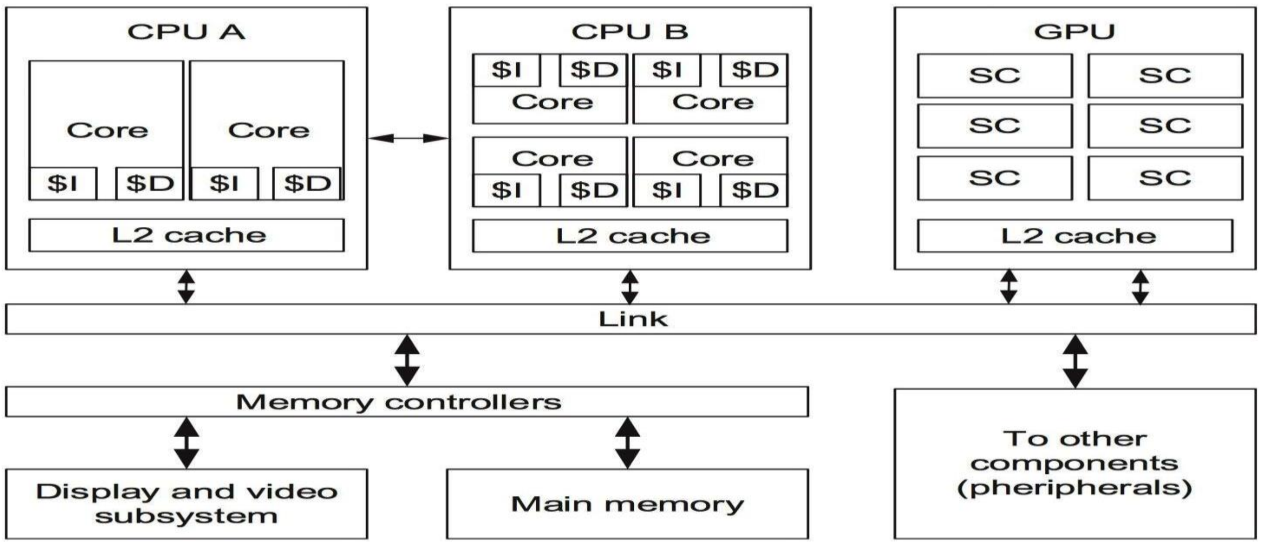

Fig: Block diagram multicore processor

Figure depicts Cores in heterogeneous multi-core systems are not all the same. Multi-core systems' cores can implement architectures like VLIW, superscalar, vector, or multithreading, just like single-processor systems can. In various application domains, including general-purpose, embedded, network, digital signal processing (DSP), and graphics, multi-core processors are frequently employed (GPU). Core counts can reach even dozens, over 10,000 for specialized devices, and over 10 million in supercomputers. The software algorithms utilized and how they are implemented have a significant impact on the speed boost provided by the utilization of a multi-core CPU. Particularly, the percentage of software that can run concurrently in parallel on several cores limits the benefits that can be made; Amdahl's law describes this effect. The best case scenario for so-called embarrassingly parallel problems is that they may see speedup factors close to the number of cores, or even higher provided the problem is sufficiently divided to fit within each core's caches, avoiding the need of much slower main-system memory. However, unless programmers spend time restructuring, the majority of apps are not as significantly accelerated. Software parallelization is a significant and active area of research. Applications for multiple processors that are co-integrated give designers of network architectures more design freedom. Systems using these protocols additionally have the capacity to adapt within parallel models

5. DETERMINING THE PARALLELIZATION FRACTION

Finding out how much quicker the program finished the task with N cores than it did with a single core is the first step in using these results to gauge a program's efficiency at parallelizing data. Simply divide the amount of time the action took with a single core by the amount of time it took with N cores to determine this. In our example, the speedup is 645.4/328.3 or 1.97 for two cores.
You must use the parallelization equation we previously provided and different values for P to determine the parallelization fraction:

$$S(n) = \frac{1}{(1-P) + \frac{P}{n}}$$

Try P=.8 (or 80 percent parallel efficient) and run this calculation for each number of cores as a decent starting point. For instance, the equation for 4 cores would be

$$S(n) = \frac{1}{(1-.8) + \frac{.8}{4}}$$

this comes to 2.5. You can see that our example program is actually more than 80% efficient by comparing this to our actual speedup in the example (which was 3.75), therefore we need to raise the parallelization fraction to something greater. The actual fraction in our situation was.97 (97%) which is a respectable result.
You need to know the operating frequency and the number of cores of the CPU you used to benchmark with and the CPU you are interested in in order to evaluate a CPU's performance. With those specifications in hand, you must first determine how many actual cores each CPU has using the following equation:

$$EffectiveCores = \frac{1}{(1-P) + \frac{P}{CPUCores}}$$

Basically, this is using the same parallelization equation we used earlier only using the actual number of cores the CPU has. This gives us the effective number of CPU cores the CPU has when running your program if the program was actually 100% efficient. From this, we can multiple the number of effective cores with each CPU's operating frequency to get what is essentially how many operations per second the CPU is able to complete (or GFLOPs):

$$GFLOPs = CPUFrequency * EffectiveCores$$

Finally, we can estimate how long it would take the CPU you are interested in to complete the same action you benchmarked by dividing the GFLOPS of the two CPUs and multiplying it by the time it took your test CPU to complete the action with all of it's cores enabled:

$$Performance = \frac{CPU1GFLOPs}{CPU2GFLOPs} * CPU1BenchTime$$

6. ADVANTAGES

- Increased computing Capabilities.
- Higher Performance.
- The digital home & Business.
- Quality software development.
- Multicore processors can finish more work than single centre processors.
- Turns out incredible for multi stringing applications.
- Can finish synchronous work as low recurrence.
- They can deal with more information than single centre processors.
- They can finish more work while burning-through low energy when contrasted with the single centre processor.
- You can do complex works like filtering of the infection against infection and viewing a film simultaneously.
- As the two centres of processors are on single chip so PC reserve exploits and information has not to travel longer.
- PCB (printed circuit board) needs less space in case of utilizing multi core processors.

7. APPLICATIONS

- Machine vision
- CAD systems
- CNC machines

- Automated test systems
- Motion control

8. CONCLUSION

Every time a new generation of a CPU is released, the chip's number of cores grows. Depending on the demand, multicore CPUs become not only quicker but also more power efficient. We must assess the CPU according to the workload we anticipate processing on these systems in order to keep meaningful profiling. We have discussed metrics, factors, benchmarking tools, and approaches for evaluating multicore CPU performance. We have also given an example of performance evaluation for multicore CPUs. Depending on the objective of the study, many methodologies are employed in multicore CPU performance analysis. As multicore CPU production reaches new heights, we anticipate seeing new methodologies in multicore CPU performance analysis. By simply adding more cores, it is feasible to boost a processor's performance without raising the clock frequency. The impossibility of using a single core processor at a high clock frequency has made multicore technology possible and has made it the current industry trend.

REFERNCES

[1] Duheon Choi, Kwangsu Kim &Eui-Young Chung, Asymmetric Prefetching Architecture for Multicore Processor Computer, vol-02, pp. 1-1.

[2] A. Roy, Jingye Xu & M. H. Chowdhury. Multi-core processors: A new way forward and challenges, International Conference on February 2019, pp 454-455.

[3] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta & P. W. Cook. Power-aware microarchitecture: design and modeling challenges for next generation microprocessors, IEEE March 2018, vol-20, pp 26-30.

[4] D. Patterson, Balajivenu. The trouble with multi-core, Spectrum. IEEE, January 2010, Vol-14, pp 28-32.

[5] V. Venkatachalam and M. Franz. Power reduction techniques for microprocessor systems, ACM Computing Surveys, December 2005, Vol-37, pp 195–237.

[6] D. Geer. Chip Makers Turn to Multicore Processors Computer, May 2019, vol-38, pp 11-13.

[7] G. Blake, R. G. Dreslinski& T. Mudge, A survey of multicore processors Signal Processing Magazine, IEEE, March 2019, vol-26, pp 36-37.

[8] J. Yan and W. Zhang, Hybrid multicore architecture for boosting single-threaded performance, In ACM SIGARCH, Vol-35, pp. March 2016, 141-148.

[9] G. Lowney, Why Intel is designing multi-core processors, In Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 2018, pp 113-113.