CHAPTER – 10

MULTICORE COMPUTERS

INTRODUCTION:

A multicore computer also known as chip multiprocessor combines two or more processors (cores) on a single piece of silicon.

HARDWARE PERFORMANCE ISSUES:

Microprocessor systems have experienced a steady, exponential increase in execution performance.

1. INCREASE IN PARALLELISM:

The organizational changes in processor design has been primarily focused on increasing instruction level parallelism so, more work can be done in each clock cycles.

2. PIPELINING:

Individual instructions are executed through a pipeline of stages so that while one instruction is executing in one stage of pipeline, another in another stage.

3. SUPERSCALAR:

Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in pipeline.

4. SIMULTANEOUS THREADING:

Register banks are replicated so that multiple threads can share the use of pipeline resources.

5. POWER CONSUMPTION:

As the number of transistors per chip rise, power requirements grow, one way to control power density is to use more of the chip area for cache memory.

SOFTWARE PERFORMANCE ISSUES:

1. SOFTWARE IN MULTICORE:

The potential performance benefits of a multicore organization depends on the ability to effectively exploit parallel resources available to the application. The Amdahl's law state that,

$$Speed\ up = \frac{time\ to\ execute\ program\ on\ single\ processor}{time\ to\ execute\ program\ on\ N\ parallel\ processors}$$

$$= \frac{1}{(1-f)+\frac{f}{N}}$$

The fraction (1 - f) involves code that is inherently serial and fraction 'f' that involves code that is infinitely parallel.

MULTICORE ORGANIZATION:

The main variable in multicore organization are as follows:

- The number of core processors on the chip.
- The number of levels of cache memory.
- The amount of cache memory that is shared.

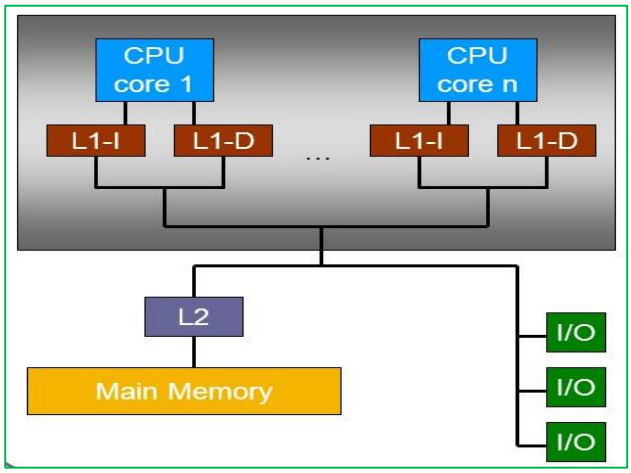

Fig (a): Dedicated L1 Cache

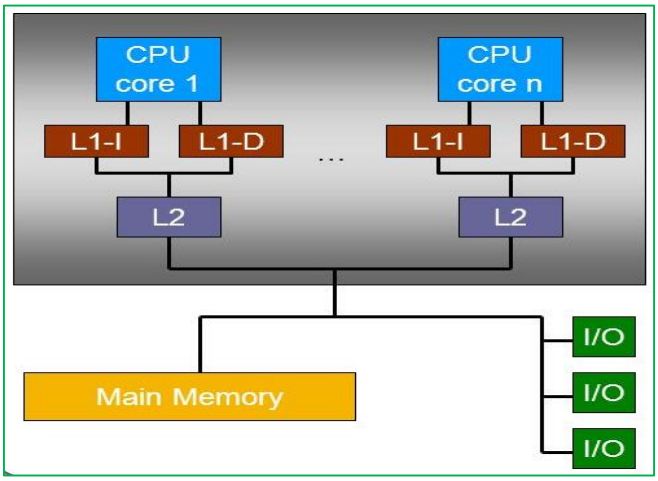

Fig (b): Dedicated L2 Cache

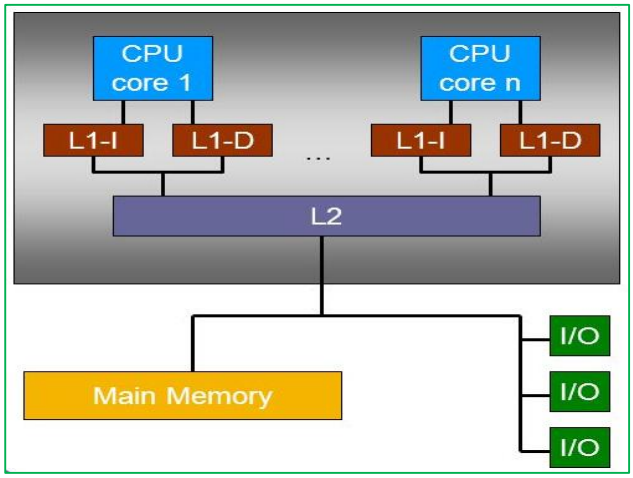

Fig (c): Shared L2 Cache

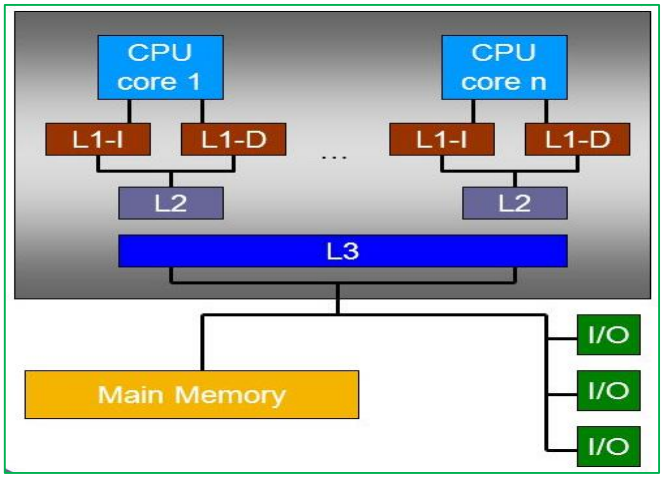

Fig (d): Shared L3 Cache

- D = Data
- I = Instruction

Figures above shows four general organizations for multicore systems.

Figure (a) is an organization found in some of the earlier multicore computer chips and still found in embedded chips. Here, only on chip is L1 with each core having its dedicated L1 cache. L1 cache is divided into instruction and data caches. For example: ARM Processors

Figure (b) is also one in which there is no on chip cache sharing there is enough area available on the chip to allow for L2 cache. For example: AMD Processor.

Figure (c) shows a similar allocation of chip space to memory but with the use of shared L2 cache. For example: Intel Core Duo

Figure (d) shows that the amount of cache memory available on the chip continuous to grow having a shared L3 cache with dedicated L1 and L2 for each core processor. For example: Intel Core i7.

DUAL CORE AND QUAD CORE PROCESSOR:

1. Dual Core Processor:

- CPU that includes two complete execution cores.
- Combines two processors and their cache controllers in a single chip.
- Operating system has sufficient resources to handle intensive tasks in parallel.
- Example: AMD Processor, Intel Core 2 Duo, etc.

2. Quad Core Processor:

- Has four independent units i.e. cores.
- Individual core can run multiple instructions at that same time.
- Has more processing speed.
- Example: Intel Core i5

POWER EFFICIENT PROCESSORS:

Power management has become a major issue in the design of multi-core chips. There are many negative effects that result from increasing power consumption such as unstable thermal properties of the die and hence affecting the system performance which makes power consumption issue sometimes more important than speed. An important observation is that threads running on different cores do not need the same power all time to execute at high performance. There are some waiting times due to memory read/write operations for example which require saving unnecessary processing power. So, to achieve a good balance between scalar performance/throughput performance and power it is essentially required to dynamically vary the amount of power used for processing according to temporal analysis of the code needs.

Developed power management techniques can be classified into two main categories: reactive and predictive.

In reactive techniques, the technique reacts to performance changes in the workload. In other words, a workload may initially have states that need high performance, others of I/O waits and low performance. When the state of the workload changes, the technique reacts to that change accordingly. However, there might be some lag between workload phase changes and power adaptation changes which may lead to states of either in-efficient energy consumption or performance degradation.

On the other hand, predictive techniques, for example, overcome this issue. Those techniques predict phase changes in the workload before they happen, and hence act immediately before a

program phase changes. That leads to optimal energy-saving and performance results. However, there is no workload that can be fully predicted, so reactive techniques are used for portions that cannot be predicted (which is usually more than 60% of the entire workload). So, reactive techniques are inevitable to use and consequently we concentrate in this study on those techniques.

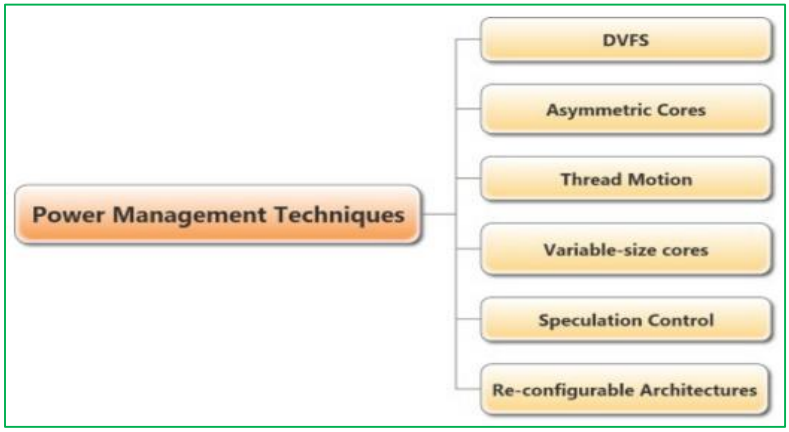

Fig: Power Management Techniques