**Memory Map: A Multiprocessor Cache Simulator**

Shaily Mittal1  and Dr. Nitin2

1 Assistant Professor, CSE Department, Chitkara University, Baddi, Himachal Pardesh, India.

2 Department of CSE & IT , Jaypee University of Information & Technology, P.O. Waknaghat, Solan - 173234, Himachal Pradesh , India.

1 shaily22mittal@gmail.com 2delnitin@gmail.com

**Abstract:**

Now a day’s Multiprocessor system-on-a-chip (MPSoC) architectures are mainly focused by manufacturers to provide increased concurrency, instead of increased clock speed, for embedded systems. However, managing concurrency is a tough task. Hence, one major issue is to synchronize concurrent accesses to shared memory. An important characteristic of any system design process is memory configuration and data flow management. Although, it is very important to select a correct memory configuration, it might be equally imperative to choreograph the data flow between various levels of memory in an optimal manner.

Memory Map is a multiprocessor simulator to choreograph data flow in individual caches of multiple processors and shared memory systems. This simulator allows user to specify cache reconfigurations and no. of processors within the application program and evaluates cache miss and hit rate for each configuration phase taking into account reconfiguration costs. The code is open source and in java.

**Keywords:** Multiprocessors, Shared Cache, Private cache, Simulator

1. **Introduction**

In the memory hierarchy, cache is the first encountered memory when an address leaves the central processing unit (CPU) [1]. It is expensive, relatively small as compared to the memories on other levels of the hierarchy, provides provisional storage that supplies most of the information requests of the CPU, due to some customized strategies that control its operation.

On-chip cache sizes are on the rise with each generation of microprocessors to bridge the ever-widening memory-processor performance gap. According to a literature survey in [2], caches consume 25 to 50% of total chip energy, while covering only 15% to 40% of total chip area. Whereas designers have conventionally focusing their design efforts on improving cache performance as these statistics and technology trends visibly indicate that there is much to be gained from making energy and area, as well as performance, front-end design issues.

Embedded systems as they occur in application domains such as automotive, aeronautics, and industrial automation often have to satisfy hard real-time constraints [3]. Hardware architectures used in embedded systems now feature caches, deep pipelines, and all kinds of conjecture to improve average-case performance. The speed and size are two concerns of embedded systems in the area of memory architecture design. In these systems it is necessary to reduce the size of memory to obtain better performance. The speed of memory plays an important role in system performance. Cache hits usually take one or two processor cycles, while cache misses take tens of cycles as a penalty of miss handling, so the speed of memory hierarchy is a key factor in the system. Almost all embedded processors have in-chip instructions and data caches. Scratch-pad memory (SPM) has become an alternative for the design of modern embedded system processors [4, 5].

Multiple processors on a chip communicate through shared caches embedded on a chip [6]. Integrated platforms for embedded applications [7] are even more assertively pushing core-level parallelism. SoCs with tens of cores are commonplace [8, 9, 10, 11] and platforms with hundreds of cores have been proclaimed [12]. In principle, multi-core architectures have the advantages of increased power-performance scalability and faster design cycle time by exploiting replication of pre-designed components. However, performance and power benefits can be obtained only if applications exploit a high level of concurrency. Indeed, one of the toughest challenges to be addressed by multi-core architects is how to help programmers expose application parallelism.

Thread level parallelism brings revolution in MPSoC [13]. As multiple threads can be executed simultaneously, it makes the real advantage of multiple processors on a single chip [14]. But this leads to a problem of concurrent access to cache by multiple processors. When more than one processor simultaneously wants to access same shared