Multithreading an Proficient Technique for Enhancing Performance of System

*Er. Isha Choudhary, **Er. Paramveer Singh Gill

*Research Scholar, M.Tech. ECE (VLSI), Chandigarh Engineering College, Landran, Ishachaudhary18@gmail.com
**Assistant Professor, ECE Dept., Chandigarh Engineering College, Landran, paramveer.ece@cgc.edu.in

Abstract
Multithreading has been shown to be powerful approaches for boosting a system performance by taking advantage of parallelism in applications. A multithreaded processor is able to pursue two or more threads of control in parallel within the processor pipeline. Multithreading results to tune the application performance considerably. To recognize the advantages of model-driven development in designing parallel applications, a survey is done and it is concluded that model driven development combined with multithreading could achieve enhanced benefits.

Keywords: - Multithreaded processors, Parallelism, Performance.

INTRODUCTION
A multithreaded processor is able to concurrently execute instructions of different threads of control within a single pipeline. The nominal constraint for multithreaded a processor is the ability to chase two or more threads of control in parallel within the processor pipeline, that is, the processor must provide two or more independent program counters, an internal tagging mechanism to distinguish instructions of different threads within the pipeline, and a mechanism that triggers a thread switch[1]. The toggling of the overhead thread must be very low i.e. few cycles only. Multithreaded processor features regularly not always, multiple register sets on the processor chip. Presently the major interest in hardware multithreading stems from two facts:
—Latency reduction is an important task when designing a microprocessor. Latencies arise from data dependencies between instructions within a single thread of control. Long latencies are caused by memory accesses that miss in the cache and by long running instructions. Short latencies may be bridged within a superscalar processor by executing succeeding, nondependent instructions of the same thread. Long latencies, however, stall the processor and lessen its performance[1].
—Shared-memory multiprocessors suffer from memory access latencies that are several times longer than in a single-processor system. When accessing a nonlocal memory module in a distributed-shared memory system, the memory latency is enhanced by the transfer time through the communication network. Extra latencies arise in a collective memory multiprocessor from thread synchronizations, which cause idle times for the waiting thread. One solution to fill these idle times is to switch to another thread. However, a thread
switch on a conventional processor causes saving of all registers, loading the new register values, and several more administrative tasks that often require too much time to prove this method as an efficient solution[1].

Explicit Multithreading
The deployment of coarser-grained parallelism by the use of technique known as explicit multithreaded processors is the solution surveyed in this paper that interleave the execution of instructions of different user-defined threads (operating system threads or processes) in the same pipeline. Multiple program counters are existing in the fetch unit and the multiple contexts are frequently stored in different register sets on the chip [2]. The execution units are multiplexed between the thread contexts that are loaded in the register sets. In the computation the latencies that arise of a single instruction stream are filled by computations of another thread. Thread-switching is performed automatically by the processor due to a hardware based thread-switching policy. This ability is in contrast to conventional processors or today’s superscalar processors, which use busy waiting or a time-consuming, operating system-based thread switch[2][3]. Depending on the specific multithreaded processor design, either a single issue instruction pipeline (as in scalar processors) is used, or multiple instructions from possibly different instruction streams are issued simultaneously. The latter are called simultaneous multithreaded (SMT) processors and combine the multithreading technique with a wideissue superscalar processor such that the full-issue bandwidth is more often utilized by potentially issuing instructions from different threads simultaneously.

Implicit Multithreading
A special approach boosts the performance of sequential programs by pertaining thread-level speculation. This type of thread in multithreaded processor is referred to as contiguous region of the static or dynamic instruction sequence. Those such processors implicit multithreaded processors, which refers to any processor that can dynamically generate threads from single-threaded programs and execute such speculative threads concurrent with the lead thread. In case of misspeculation, all speculatively generated results must be squashed. Threads generated by implicit multithreaded processors are mostly executed speculatively in contrast to the threads in explicit multithreaded processors[2][3].

Related Work
Miao Ju et. al. presented a chip multiprocessor performance (CMP) investigation method. A CMP is modeled generically at the thread level, overlooking the instruction-level and micro architectural details. The aim is to allow various possible program-task to core mapping choices to be tested quickly in the initial programming phase, when the executable program is yet to be developed[4]. On the basis of this methodology, the analytical modeling technique based closed queuing network models and the simulation tool were developed. Both approaches were found to be fast and accurate, allowing fast performance testing
of CMPs with large numbers of cores and threads in the initial programming phase. Ran Zhang and Hui Guo investigated the energy efficiency of varied multi-threaded processor designs. Based on a six-stage PISA processor, it was shown that in terms of potential energy saving the coarse-grained design is better than the fine-grained design[5]. Furthermore, for the coarse-grained design, the thread number for the optimal energy efficiency is closely related to the memory access delay. When the memory access latency is small, the low-thread processor appears more energy efficient; When the memory delay increases the high-thread processor becomes superior. Jian F et. al. observed that threads are usually short code sequences with no branches and few memory side-effects, which means that the number of checkpoints is small and constant. In addition, the state structures of a thread already presented in hardware can be reused to provide checkpointing. They demonstrated this principle of using a hardware/software co-design called Rethread, which features compiler-generated code annotations and automatic recovery in hardware by restarting threads[6]. This method offers the ability to recover from transient faults without the using special hardware. Moreover, results show performance degradation under both fault-free condition (<5%) and as a function of fault rate. Mahanama Wickramasinghe and Hui Guo proposed a thread scheduling algorithm that exploits the execution behaviours of given application threads and dynamically schedules them based on their relative real slack times to balance the thread execution so that the processor resource can be maximally utilized and all threads can be completed within their deadlines with the minimal overall execution time. To further reduce the execution time, they presented a thread scheduling control design where the thread selection and execution switching are performed in parallel and the execution switching logic is inserted into the non-critical path in the pipeline so that the thread switching takes zero clock cycles. In their experiments on a set of tests show that about 50% time savings can be achieved as compared to the traditional round robin scheduling approach without incurring high energy consumption levels. Joseph M. Arul et. al. presented the implementation of SDF architecture using VHDL as well as two ways of the passing of data in a multithreaded architecture. The hardware consumption was very less. By using two different methods of the passing of data, i.e., RTR and RTM methods, the difference between these two methods are observed clearly. The RTR method is more efficient, but the usage of register context could consume more power as shown in section four. It is explored that this architecture for embedded system with multi-core. Mahanama Wickramasinghe and Hui Guo presented with the aims to reduce instruction cache misses in a single pipeline processor for applications that offer embarrassing parallelism and enable the same code to be executed by a number of independent threads on different data sets. Such a design can be used as a building block processor for large multicomputer systems[7]. They proposed a micro-architectural level multithreading control design, which synchronizes the thread execution to allow cached instructions to be maximally reused by all threads. Their experiments show that our design not only increases the pipeline performance but also reduces the memory access frequency, hence effectively achieving high energy.
Mohamad Hafezan and Leila Beigi proposed a number of analytical expressions to consider multithreading and branch predictor parameters in the model, resulting in having a more accurate representation with more effective design parameters. The multithreading is a selectable option in design of network processors which depending on the application can be enabled or disabled. Using the proposed model they not only can find out how to optimally configure a network processor but also can understand if the multithreading helps to improve the performance [8]. Through simulation they have also observed a significant fraction of dynamic power (3% to 5% of total power consumption) is consumed by the branch predictor access, therefore an the branch predictor parameters was added to the model to consider its impact on the performance. Validity of the proposed model has been verified as performance trends for both of the simulation and proposed model were similar and the average error was equal to only 0.04.

Conclusion

A multithreaded processor is capable to perform or process two or more threads of control in parallel within the processor pipeline. Multithreading results to tune the application performance considerably. To recognize the advantages of model-driven development in designing parallel applications, a survey is done and it is concluded that model driven development combined with multithreading could achieve enhanced benefits. Latency, transient faults, power consumption, memory access frequency are the major concerns which are needed to be addressed with respect to the multithreaded processors.

References:

