IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249-9555
Vol. 2, No. 1, 2012

Study on Data Prefetching techniques for Linked Data Structure Applications

J.Saira Banu, Assistant Professor,
School of Computing Science & Engineering,
Vellore Institute of Technology,
Vellore, India.

Abstract— **Data Prefetching is a promising solution to hide memory access latency. An application with Linked Data Structure (LDS) aggravates *the* memory access latency problem. There are quite number of research publications to solve this problem. Helper thread prefetching is a method in which the cache miss of an application is predicted by a thread ahead of actual program execution and masks the memory latency problem. LDS are commonly found in compilers, databases and graphics applications. This paper explores prefetching techniques to improve the performance of LDS applications in Chip Multiprocessor (CMP). A comparative study of various techniques with helper thread prefetching technique is also examined in this paper.**

***Keywords-** Helper thread, Linked Data Structure, Latency, performance*

I. INTRODUCTION

With the development of multicores, the processing speed has increased albeit with an extended gap between the processor speed and memory speed. Memory wall problem, memory bandwidth and slow I/O access speed are the limiting factors for memory performance in multicores. Memory wall problem is due to high memory access latency. Caching and prefetching are the commonly used methods to bridge the gap between computing and memory performance. By utilizing an idle core within a multi-core processor, helper thread prefetching improves the existing gap between processor and memory speed. Helper thread prefetching helps in improving the memory performance of a Chip Multiprocessor (CMP). Linked Data Structures (LDS) are broadly used in compilers, databases and graphics applications. Helper thread prefetching forms a viable solution to improve the performance of LDS application programs. Helper threads are also known as assist threads which executes in parallel with the main thread. It improves the performance of the main thread by prefetching the delinquent loads of the main thread which causes cache misses. It should be ahead of the main thread to ensure timely prefetching of data. The rest of the paper is organized as follows, Section 2 discusses the latency hiding technique via prefetching , Section 3 describes Prefetching techniques for LDS applications, Section 4 deals with helper thread prefetching for LDS applications in Simultaneous Multithreading (SMT), Section 5 deals with helper thread prefetching for CMP, Section 6 provides Conclusion.

II. LATENCY HIDING TECHNIQUE VIA PREFETCHING

Processor speed is increasing at much faster rate than memory and interconnection networks. Thus in scalable multiprocessor or large-scales multi-computer there is a need for latency- reducing technique. The existing methods to reduce memory latency problem are prefetching, coherent caches, relaxed memory consistency models and multiple contexts. Prefetching reduce the memory latency problem by bringing the data closer to the processor cache before needed. In coherent caches a special hardware is employed to reduce the cache misses. Through buffering and pipelining of memory references, relaxed memory consistency models removes the memory latency problem. In Multiple contexts approach long latency operations are handled by switching of processor from one context to other. The basic approach in single-core or multi-core processor to hide memory access latency is by adopting a multi-level memory hierarchy such as cache memory. Memory access time is reduced by the cache memory if the required data is found in the nearest storage. If processor required data is not in the cache it leads to cache misses which still causes memory access latency. Inspite of continuous advances in cache design and development of new prefetching techniques, the memory latency problem continue and aggravates especially with pointer-intensive applications, which display poor spatial and temporal locality. Most prefetchers are history-based: they analyze data access patterns performed in the past, predict that future data accesses will follow similar patterns, and prefetch the corresponding data. While this approach works for programs with regular data access patterns, such as array based scientific programs. It is not effective for programs whose data accesses depend on the input or not structured in easily predicted patterns, or do not contain recurrences (that is, frequent reuse of the same remote data). Because LDS consist of dynamically connected and allocated nodes generally taking the form as trees, graphs, and lists, the address of the prefetching node must be obtained from an adjacent node, and the prefetching tends to achieve little overall performance gains. Five fundamental issues in designing a prefetching strategy are what to prefetch, when to prefetch, what is the source of prefetching, what is the destination of prefetching and who initiates prefetching. The desired characteristics of a prefetching mechanism are (1) accuracy: make a good prediction about what remote data will be accessed by the program, (2) effectiveness: make the remote data available locally by the time the program needs it, and (3) correctness: guarantee that prefetching does not affect the program's behavior. Two major categories of prefetching applicable to both single-core and multi-core processors are software controlled and hardware controlled. Software-controlled prefetching enables a programmer or a compiler to insert prefetch instructions into programs. An important issue with software based prefetching is that since it occurs at compile time, there is no way to identify whether the prefetched data was removed from the cache before it was utilized, and cannot therefore issue another prefetch compensate it. Hardware prefetching is specifically concerned with prefetching algorithms implemented in a dedicated hardware without software support. Hardware prefetching techniques can operate on both data and instruction caches. Software techniques normally prefetch data. Hybrid hardware/software-controlled strategies are used with multi-thread support. It requires a hardware support to run threads that are specifically executed to prefetch data and software support to synchronize the prefetching thread with the actual computation thread [6]. Table 1 shows the broad classification of Prefetching types with associated applications.

TABLE1: BROAD CLASSIFICATION OF PREFETCHING

Prefetching Type	**Methods**	**Used in applications**
Hardware Controlled	1.History based 2.Run ahead Execution based 3.Offline analysis	With regular data access patterns. with regular or irregular patterns with repeating function calls with loops or for those with same access patterns repeating in each run of an application.
Software Controlled	1.Compiler controlled 2.Application Function Calls 3.Post-Execution Analysis	with loop codes
Hybrid (Hardware/Software) Controlled	1. History based 2.Pre- execution based	with regular data access patterns With regular and random accesses

III. Prefetching techniques for LDS

Scientific applications with large array-based operations are suitable for hardware and software prefetchers. These applications exhibit streaming memory access patterns which are predicted and prefetched by the hardware and software prefetching techniques effectively. Data intensive applications such as medical imaging, weather forecasting, and astronomical data analysis are examples for LDS which exhibits irregular data access patterns. Data intensive applications have long been suffering from the relatively slow speed of main memories compared to the performance of current CPUs.

There are numerous methods proposed to improve the performance of LDS program. Roth et al (1998), proposed a dependence based prefetching method by exploiting the dependence relationships between loads that produce addresses and loads that consume these addresses. This is used to identify the regularities in the address generated rather than in the addresses themselves. Dependence-based prefetching masks the capacity misses without explicit overhead. Pointer dependences in LDS are easy to find with the dependency based approach but this is not successful for other data structures, sparse matrices and index trees for instance, whose traversal does not yield address sequences with arithmetic properties [2].Roth and Sohi (1999) proposed the Effective Jump-Pointer prefetching (JPP) method which uses a jump pointer along with the normal pointer known as chain pointer for LDS prefetching. This method combines chain and jump pointer to form four different prefetching idioms implemented in hardware, software and cooperative techniques. Cooperative technique is advantages when compared to hardware and software techniques in terms of performance and cost. It provides better performance with reduced implementation. JPP is effective when limited work is available between successive dependent accesses to enable aggressive scheduling techniques to prefetch effectively [3]. Library-based Prefetching (LBP) proposed by Malhotra & Kozyrakis employs a prefetching code inside the data structure library code. The user application is not modified in this approach with different hardware since the prefetching code is embedded in the library. The significant advantage of this method is it removes the burden on processor design and verification compared to hardware approach. LBP does not require extensive pointer analysis capabilities. Furthermore, LBP works well with varying hardware parameters (cache sizes, cache and memory latencies).Compared to pre-execution based techniques, no profiling information is required, hence it is easier to use on a larger variety of systems and datasets. Tuning of prefetch thread for every application is eliminated in this approach. The shortcomings of LBP are, it utilizes the whole CPU and makes it difficult to use in parallel application, limited to common data-structures and regular access patterns [4].Having a study on these three methods, it is clear that these methods hide the memory latency of LDS programs by exploiting dependencies, by adding jump pointers in addition with chain pointers and by using library based approach. All these methods are targeted towards LDS programs and not intended to other data structures, such as sparse matrices and index trees whose address sequence doesn't show arithmetic properties.

IV. HELPER THREAD PREFETCHING IN SMT

Simultaneous multithreading (SMT) is a technique used to improve the efficiency of a uniprocessor system with hardware multithreading. In a SMT uniprocessor, a slice of the program is extracted for each critical instruction and it forms the helper thread. Helper threads in SMT are beneficial for improving the performance of a system with small chunks of data because they minimize the interference between the helper thread and the main thread in the L1 cache. Helper threads in a SMT system can run either in a logical processor or in the same core where actual execution of the code takes place. Lee et al. clearly distinguish the helper thread prefetching in SMT system with CMP by finding certain limitations such as resource contention, synchronization overhead, complex thread management support, not suitable for applications with irregular data access patterns and inlined prefetching. In a CMP, the number of cores is integrated into a single chip with a common memory module. Helper threads in CMP overcome prefetching techniques are advantageous in handling LDS applications compared to hardware, software and other conventional prefetching techniques. Cache trashing problem, Overhead of memory access and improving timeliness of prefetching can be achieved with the help of SKIP helper thread. By overcoming these problems, the performance of an LDS application is improved. Helper threads in CMP are executed either in idle core or in the memory subsystem. Idle core in a CMP are used in various approaches such as dual core execution approach to construct a large distributed instruction window and future execution approach to pre-execute future loop iterations using value prediction to speed up single threaded programs. Dedicating an idle core for prefetching thread in high computing environment is not possible because every core in CMP is utilized for the computation task. To address this issue, memory side prefetching is proposed with an intelligent memory processor residing within the memory. The intelligent memory processor has heterogeneous mix of processors such as main processor with high memory access latency and memory processor with small memory latency. The main processor is dedicated for application program and the memory processor is used for helper thread execution. This approach can improve the performance of LDS application parallel workloads. However, scalability is a limitation of memory processor which can be overcome by a server based push method by using server cores [6]. These points clearly specify the need of memory processor to improve the performance of parallel workloads. OpenMp and MPI are used to parallelize the application program to improve the execution speed. Furthermore to improve the performance of parallel workloads, helper threads run on a memory processor and processors.

synchronization overhead and drawbacks of inlined prefetching. It is also suited for applications with irregular data access patterns and it doesn't require hardware for dynamic optimization. [1].These points illustrate the disadvantages of helper thread prefetching in SMT systems and how it is handled in CMP. This shows helper thread prefetching in CMP is more advantageous than SMT to handle LDS applications which exhibits irregular data access patterns.

V. HELPER THREAD PREFETCHING IN CMP

The Problems in Existing Prefetching techniques for LDS are large number of useless prefetches which degrades the performance, hardware overhead for storing pointers and resource contention in memory system Conventional Helper thread prefetching such as Ahead Prefetching (AP) and Active Threaded Prefetching (PV) suffers through various problems such as helper thread should not be too far from main thread, Synchronization overhead and resource contention problem. According to Huang et al SKIP helper thread prefetching techniques significantly improves the performance and reduces the resource consumption in CMP compared to conventional helper thread prefetching methods. [5]. These points exhibit that SKIP helper thread

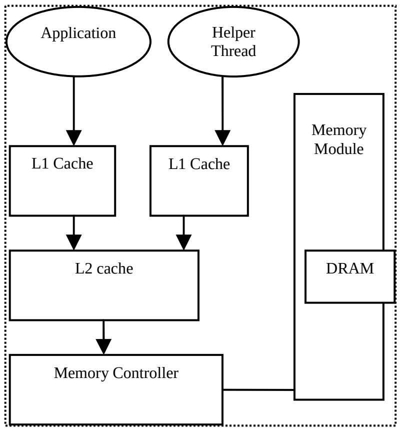

Fig1 Helper thread prefetching in idle core

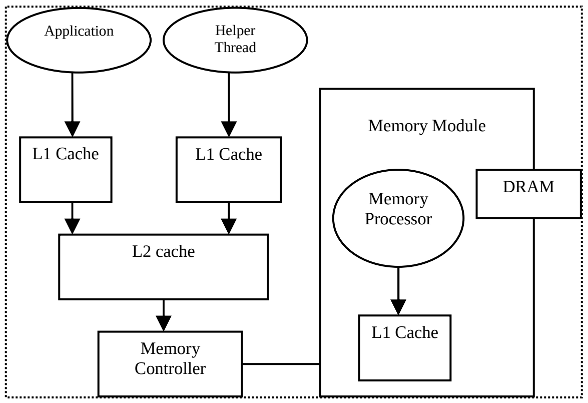

Fig 2: Helper thread prefetching in memory processor

VI CONCLUSION

Helper thread prefetching has been identified as a viable solution for lessening the gap between memory speeds and processing speed in LDS application programs. Helper thread prefetching overcomes the disadvantage of programs. With the help of parallel programming languages many other methods used for prefetching LDS application helper threads running in a memory processor. Thus, helper thread prefetching has improved the performance of LDS application programs.

REFERENCES

[1]. Jaejin Lee; Changhee Jung; Daeseob Lim; Yan Solihin; (Sept 2009), "Prefetching with Helper Threads for Loosely Coupled Multiprocessor Systems," Parallel and Distributed Systems, IEEE Transactions on, vol.20, no.9, pp 1309-1324, doi: 10.1109/TPDS.2008.224.

[2]. Amir Roth, Andreas Moshovos, and Gurindar S. Sohi; (1998), “Dependence based prefetching for linked data structures”. SIGOPS Oper. Syst. Rev. 32, 5 (October 1998), 115-126, doi: 10.1145/384265.291034 .

[3]. Roth, A.; Sohi, G.S.;(1999), "Effective jump-pointer prefetching for linked data structures," Computer Architecture 1999. Proceedings of the 26th International Symposium on, pp.111-121, 1999 doi: 10.1109/ISCA.1999.765944.

[4]. Varun Malhotra and Christos Kozyrakis; (Feb 2006), “Library-based Prefetching for Pointer-intensive Applications”, Computer Systems Laboratory, Stanford University.

[5]. Yan Huang, Jie Tang, Zhi-min Gu, Min Cai, Jianxun Zhang and Ninghan Zheng; (June 2011), “The Performance Optimization of Threaded Prefetching for Linked Data Structures”, International Journal of Parallel Programming, pp.1-23,doi: 10.1007/s10766-011-0172-7.

[6]. Byna S, Chen Y, Sun XH ;(May 2009), “Taxonomy of data prefetching for multicore processors”. Journal of Computer Science and Technology 24(3): pp 405-417 May 2009.

AUTHORS PROFILE

Saira Banu is working as an Assistant Professor in Vellore Institute of technology. She is currently pursuing her PhD degree in the area of parallel computing. Her areas of interest are computer architecture, parallel computing, and parallel algorithms. She is a life time member of CSI.