A Survey on Helper Threads

Rashmi Hegde and Gautham Katta

Abstract:

Helper Threading is a technique to accelerate a program by exploiting a processors' multithreading capability to run "assist" threads. Data prefetching, specially targeting **delinquent loads** in order to avoid memory latency and useless speculation is the motivation of various helper threading mechanisms implemented on different platforms. This paper conducts a **survey on various helper threading schemes** that are implemented on variety of platforms from single processor to multicore processors, SSMT based simulator environment and loosely-coupled multiprocessors. This gives a broader picture of the impact of helper threads on the system performance and reducing memory latency problems.

Introduction:

Memory latencies dominate the performance of many applications on modern processors. This problem only worsens as CPU clock speeds continue to advance more rapidly than memory access times, and as the data working sets and complexity of typical applications increase. **Helper Threading (HT) is one of the hardware technique used to alleviate this memory bottleneck.**

HT masks the cache miss latency at the thread level. In simplest terms, a single-threaded process is running on a multithreaded processor. In advance of a series of **heavy loads referred to as delinquent loads**, the process spawns a helper thread and passes it all necessary live-in variables. Given all necessary information, the helper thread performs only the delinquent load operation. If the two threads are properly synchronized, the main thread will reach the delinquent load to find that all values have been preloaded into the cache. As a result of helper threading, the total execution time is greatly decreased, because the negative performance impact of the delinquent load has been greatly reduced by the advanced work of the helper thread.

The remaining part of this survey paper illustrates HT and the various implementations of HT. The paper is structured as follows- *Brief overview* of all the papers considered for this survey; *Main idea and concepts* which explains the challenges involved state-of-the-art; *Future Work* and *Conclusion.*

Brief overview:

To achieve high performance, contemporary processors systems exploit two levels of parallelism: **Instruction Level Parallelism (ILP) and Thread Level Parallelism (TLP)**. Processors exploit ILP by executing multiple instructions from a single program in a single cycle. TLP is exploited by executing different threads in parallel on different processors or on a single processor [1]. Simultaneous Multithreading (SMT) [2] is a technique that combines the hardware features of wide-issue Superscalar machines and Multithreaded processors. SMT processor selects instructions for execution from all threads which can issue, thereby exploiting ILP. SMT uses ILP and TLP to substantially increase effective processor utilization and to

accelerate both multiprogramming and parallel workloads. HT are also run on SMT to further increase the performance.

Helper Threading improves the performance of a multithreaded processor by **running subordinate threads**. It uses otherwise idle hardware thread contexts to execute speculative threads on behalf of the main thread. These speculative threads attempt to trigger future cache miss events far in advance of access by the main thread to avoid the memory miss latency. **HT targets load instructions that have unpredictable access patterns and chains of dependent loads**. These load instructions are called as delinquent loads.

Virtual Multithreading (VMT) is a form of switch-on-event user-level multithreading that monitors dynamic program execution for particular targeted events. VMT is used for processors that do not have built-in hardware support for multithreading [3]. **VMT helper threads easily adapt to changing execution** domains both in terms of different hardware platforms and changing program input sizes.

The Itanium 2 processors are **single-threaded** and are not capable of multiplexing execution resources between the main thread and the prefetching helper threads except through OS-based context swapping and synchronization. Itanium 2 processors prototyped using VMT run 2 helper threads to target delinquent loads to achieve significant speedups. The throughput performance of highly threaded workloads can also be improved by reducing the **latency of individual threads using HT** [4].

HT is implemented through **dynamic optimization on the latest UltraSPARC Chip Multiprocessing (CMP)** [5]. Helper threads are generated dynamically thereby adapting to the run-time behavior of a program due to change in input data. Here, CMP uses 2 identical cores where the **application runs on one core and the helper threads run on another** under-utilized secondary core to prefetch data. Efficient synchronization between the helper and the main threads and the impact of using different prefetch instructions in the helper threads are well illustrated.

Helper management policies for single and multi-core (CMPs) configurations are discussed [6]. Reducing the area spent to implement redundant functionality and the potential of optimization through dynamic resource allocation to reduce power are the benefits of sharing. Helpers are divided into those that are accessed on-demand and that are always active. Sharing on conjoined cores for on-demand benefits from turn-based approach, always active helpers use counter-guided method to more effectively share a pool of common helpers on multiple cores.

Helper thread prefetching scheme is designed for loosely coupled processors such as **CMP** and an Intelligent Memory System having an advantage where in **threads do not contend for processor or L1 cache** [7]. Instead of extracting a p-slice for delinquent loads, the helper thread here extracts large loop-based code section. Such **large granularity** helps to decrease the overheads of thread communication and management. A new synchronization mechanism which exploits loop-iterations is devised here.

The idea of using available cores in a CMP as **helper engines for individual threads running on active cores** thereby improving the performance of individual program threads is explored [8]. To accomplish this, **event driven helper threading (EDHT)** which uses light-weight hardware support like **hardware prefetchers** for efficient event communication is used. Helper threads are launched when a helper core receives a special event trigger from a conventional core running on non-helper mode. Various prefetching algorithms for hardware prefetchers are described here.

Three helper threading scenarios are implemented to prefetch the delinquent loads into L2 cache on the **SSMT** (Simultaneous Subordinate Multithreading) simulator [9]. The 3 scenarios are – a **static loop-based helper process, a static loop-based helper thread and a sample-based helper thread**. The comparison of these scenarios give a picture of how L2 hits are significantly increased while L2 misses are reduced and also an insight into the future work is well put across.

***Main idea and concepts**:*

SMT [2] attacks multiple sources of wastes in wide-issue processors without sacrificing single-thread performance. To enhance processor performance, exploiting parallelism is necessary. Superscalars provide low ILP, placing multiple superscalar processors on a chip is not effective as performance suffers due to low TLP. SMT consumes both ILP and TLP using the hardware resources more efficiently and hence increasing the throughput.

VMT [3] [4] **virtualizes a uniprocessor architecture which lacks in-built hardware support for multithreading** to support multiple concurrent user-level thread contexts. Resource contention among different threads poses latency issues. Self throttling mechanism to achieve inter-thread synchronization in helper threads using trigger events, fly-weight context switch and automatic thread generation provides significant performance boost even for a diverse-set of real workloads.

SMT and VMT pose resource contention problems. Most techniques rely on hardware support to reduce context switch overhead between the threads and the construction of helper threads depends upon static feedback profiling. To overcome these issues, **dynamic helper thread prefetching** on a CMP with helper threads running on one core and the application on another is used. State-of-the-art includes piggyback on the optimizer thread, dynamic optimization of control flow and synchronization for helper threads [5].

Deeply pipelined aggressive cores need to be replaced by large amount of performance accelerating hardwares due to technology scaling trends. With CMPs and more of processor being broken into helpers, reactive and always active resource sharing and resource allocation needs to be addressed. **Decoupling helpers on cores** (like helpers for D/I caches, BBTB, Data Prefetcher and Value Predictors) **to dynamically tune processors on an application phase basis** using **counter-guided helper management** is the state-of-the-art discussed here [6]. Constructive sharing and power savings from flexible sharing, improving multicore workloads using counter-guided approach are being explored.

Due to a tightly-coupled system design, thread granularity is small resulting in high thread management overhead and also the prefetching threads are not synchronized well in a loosely-coupled system. To overcome this, **loop-based extraction algorithm** which extracts a p-slice for a large loop-based code section is used. Light-weight **general semaphores** are used between the threads in a CMP while 3 special registers in a memory processor is used in an Intelligent Memory System to achieve efficient synchronization [7].

Contention of resources and no improvement in the performance of individual computation threads has led to exploring this idea of using the **available cores in a CMP as helper engines** for individual threads on active cores. EDHTs on a baseline hardware implementation (prefetcher here) are observed to give speedups of around 5%-100% over wide range of applications. Prefetching algorithms like global and local stride, DFCM (Differential Finite Context Method), Markov and Correlation prefetcher used to emulate hardware prefetching mechanisms for EDHT threads and the comparison with the recently proposed FE (Future Execution) and DCE (Dual Core Execution) affirms the efficiency of EDHTs [8].

Several scenarios that explore helper threading mechanisms on a SSMT simulator is analyzed [9]. Helper process with loop-based synchronization requires efficient communication between main and helper. Helper thread with loop-based synchronization fixes the thread creation issues and hence **eliminates the need for inter-process communication**. In Sample-based helper thread triggering, tight loop due to lack of sleep system call results in high increase in total instruction count.

Future Work:

Implementing VMT on various architectures and exploring more efficient hardware implementations and compiler optimizations to support VMT. In-thread prefetching is also one of the cache optimization techniques that benefit on chip cores. An arbitrator to choose between in-thread and helper threaded prefetching needs to be explored. Also thread synchronizing mechanism requires further evaluation on CMP.

Helper management policies need to be studied in detail as the trends in technology is scaling to new heights. EDHT implementation in conjunction with a baseline hardware implementation in order to decrease memory latency with an optimal prefetching algorithm needs to be further developed. In paper [9], all the three scenarios need to be addressed further and requires lot of improvisations to conclude on a better implementation methodology which enhances the helper threading mechanism.

Conclusion:

In this paper, we discuss Helper Threading mechanism and the implementation of Helper threads on different platforms like Itanium 2, Sun UltraSPARC, CMPs and SSMT based simulators in order to reduce memory latencies and enhance the performance of the systems.

As the gap between the processor speed and the memory latency continues to widen helper threads can become even more important, improving both latency performance and throughput performance. One of the ways of achieving helper threading is via Virtual Multithreading. Utilization/sharing of resources to improve the performance of individual threads through EDHT's and various hardware prefetching algorithms are explored.

Helper management policies for both single and multicore configurations for improving performance and power efficiency by decoupling helpers on application basis and counter-guided mechanism is another area of study. A novel technique to tackle the synchronization between the application and the helper thread and the prefetching schemes for helper threads on loosely-coupled processors is presented.

This survey on Helper Threads aims at understanding different implementations of HT on various platforms and the advantages of using these to reduce memory latencies in the memory hierarchy and increase in the overall system performance which could not be achieved by just exploiting ILP or even SMT. Different approaches for implementing helper threads have been brought out and the performance evaluation for these support the claim that HT achieves better efficiency.

References:

[1] Jack L. Lo, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Rebecca L. Stamm and Dean M. Tullsen. "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous Multithreading". ACM Transactions on Computer Systems, Vol. 15, No. 3, August 1997.

[2] Jack L. Lo, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Rebecca L. Stamm and Dean M. Tullsen. "Simultaneous Multithreading: A Platform for Next-Generation Processors". IEEE Micro, 1997.

[3] Perry H. Wang, Jamison D. Collins, Hong Wang, Dongkeun Kim, Bill Greene, Kai-Ming Chan, Aamir B. Yunus, Terry Sych, Stephen F. Moore and John P. Shen, Intel. "Helper Threads via Virtual Multithreading". IEEE Computer Society, 2004.

[4] Perry H. Wang, Jamison D. Collins, Hong Wang, Dongkeun Kim, Bill Greene, Kai-Ming Chan, Aamir B. Yunus, Terry Sych, Stephen F. Moore and John P. Shen, Intel. "Helper Threads via Virtual Multithreading On An Experimental Itanium 2 Processor-based Platform". ASPLOS '04, ACM.

[5] Jiwei Lu, Abhinav Das, Wei-Chung Hsu, Dept. of Computer Science and Engineering, University of Minnesota; Khoa Nguyen, Santosh G. Abraham, Scalable Systems Group, Sun Microsystems Inc. "Dyanmic Helper Threaded Prefetching on the Sun UltraSPARC CMP Processor". Proceedings of 38th Annual IEEE/ACM International Symposium on Microarchitecture, 2005.

[6] Anahita Shayesteh, Glenn Reinman, Norm Jouppi, Tim Sherwood, Suleyman Sair. “Improving the Performance and Power Efficiency of Shared Helpers in CMPs”. CASES ’06, October 2006, ACM.

[7] Changhee Jung, Daesob Lim, Jaejin Lee and Yan Solihin. “Helper Thread Prefetching for Loosely-Coupled Multiprocessor Systems”.

[8] Ilya Ganusov and Martin Burtscher. “Efficient Emulation of Hardware Prefetchers via Event-Driven Helper Threading”. PACT ’06, September 2006, ACM.

[9] Kurniadi Asrigo, Chris Comis, Hassan Shojania. “Implementation and Analysis of Helper Threads with SSMT”.