http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (9): 1673-1684, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1673.1684

Parallel Techniques of the Sequential Codes Based on Multi-core

[1,2]Xiang Li and [1]Jing Zhang
[1]School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, 710048, China
[2]School of Electrical and Information Engineering, Shaanxi University of Science and Technology, Xi'an, 710021, China

Abstract: Multi-core processors are becoming ubiquitous with the continuous development of hardware technology. But many applications are sequential applications and they receive no benefits running on multi-core processors. Addressing this problem, using parallel techniques improve the sequential program running on the multi-core processors. In this paper, we introduced three parallel types of thread level parallelism. Typical DOALL, DOACROSS, DSWP and PS-DSWP techniques are described. These techniques can explore the parallel from sequential application, but much dependence is not easily predictable or manifests them infrequently by the non-speculative transformation. So many speculative techniques, such as thread level speculation (TLS), Speculation DSWP (SpecDSWP), Speculative PS-DSWP (SpecPS-DSWP) and Interprocedural SpecPS-DSWP (iSpecPS-DSWP), are proposed to break problematic dependences to enhance parallelism. We introduced these speculative parallel techniques and described their execution. SpecDSWP, SpecPS-DSWP and iSpecPS-DSWP are compared from supporting speculation types, memory version and implement steps. At last, some extended TM systems which support TLS techniques are analyzed from thread spawning mechanism, context passing mechanism and sequential ordering.

Key words: Multi-core, speculative parallel techniques, thread level speculation, decoupled software pipelining, transactional memory

INTRODUCTION

Multi-core processors are becoming ubiquitous with the continuous development of hardware technology. The number of cores per chip is expected to rise in future, such as Intel's 4-core Core i7, IBM 8-core Power 7, AMD's 16-core Interlagos, Tilera's 64-core Tile 64 and Intel's 80-core Teraflops processors. Multi-core processors can improve system throughput and speed up multi-threaded applications. It can be seen from the Fig. 1 that many important applications are single threaded and single-threaded applications receive no benefits running on multi-core processors (Vachharajani *et al*., 2007). Therefore, how to using the hardware parallelism of multi-core processors to speed single applications is a challenge. Since loops of sequential applications are the largest source of parallelism considerable attention on multi-core processors (Hurson *et al*., 1997; Zhang *et al*., 2012a). This study presents and analyzes several thread partitioning techniques based on multi-core and some extended TM systems which support TLS techniques.

BACKGROUNDS AND RELATED WORK

Parallel types of thread-level parallelism: Independent Multi-Threading (IMT), Cyclic Multi-Threading (CMT) and Pipelined Multi-Threading (PMT) are three categories techniques for extracting parallelism from loops of TLP. IMT transformation prohibits cross-thread dependences and has restricted iteration level parallelism. The typical IMT transformation is DOALL technique which usually used to parallelize loops in array-based scientific programs. CMT transformation which is suitable for general purpose applications allows cross-thread dependences and maintains correct execution dependences among threads by synchrony. DOACROSS technique is the major technique in this category. PMT also allows cross-thread dependences but it only allows dependence of one direction. The first proposed PMT technique is DOPIPE (Davies, 1981) which is restricted to only loops with limited control flow. Decoupled Software Pipelining (DSWP) (Bridges, 2008) is a more general PMT technique to extract pipelined parallelism from loops with arbitrary control flow. It can be seen the difference of DOALL, DOACROSS and DSWP techniques from the Fig. 2.

Dependence graphs: There have been several ways to perform parallelism in a sequential application. The first way is Data Dependence Graphs (DDG), which adopt a node to represent the execution of an instruction and adopt edges to represent the dependences between

Corresponding Author: Xiang Li, School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China Tel: 13991185241

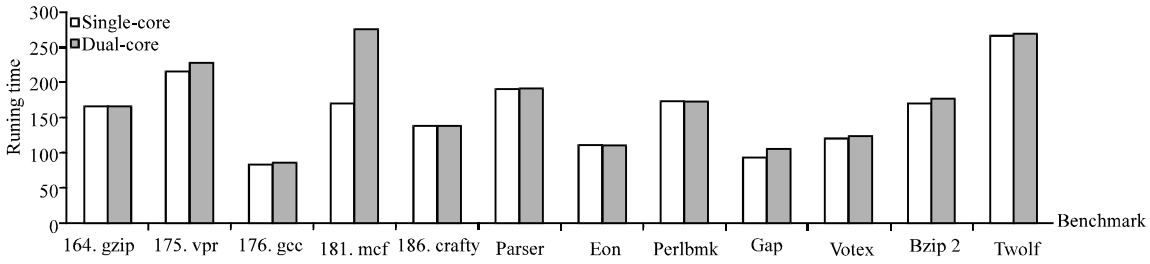

Fig. 1: Comparison SPEC CINT 2000 running time in single-core and dual-core

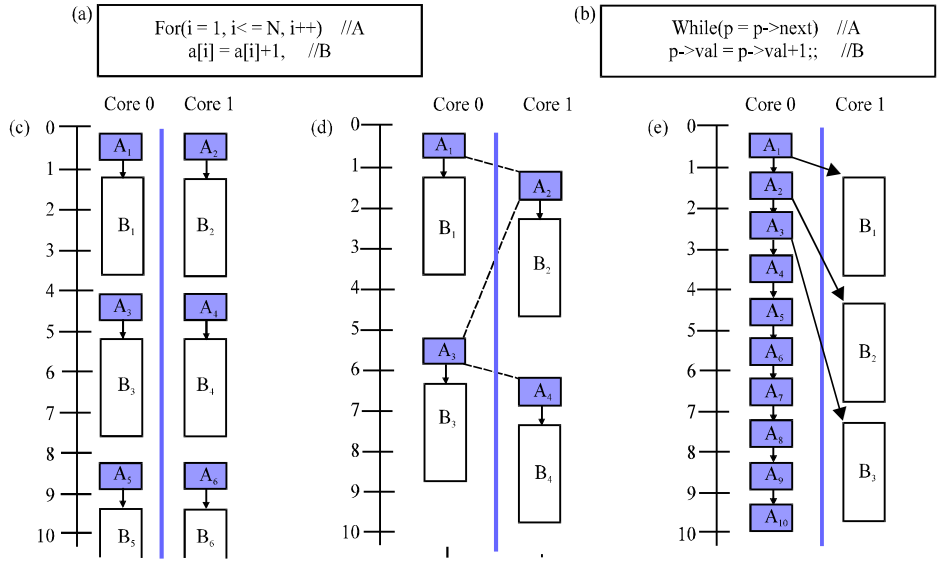

Fig. 2(a-e): Execution of DOALL, DOACROSS and DSWP Techniques, (a) Array operation, (b) Linked list operation, (c) DOALL executea, (d) DOACROSS execute b and (e) DSWP execute b

instructions. The second way is Control Dependence Graphs (CDG), which is the representation for the control flow relationships of an application. The third way is Program Dependence Graphs (PDG), which represents the loop of DSWP techniques by including both data and control dependences for each operation in an application (Ferrante *et al*., 1987).

TRADITIONAL THREAD PARTITIONING TECHNIQUES BASED ON MULTI-CORE

Thread-level program parallelization is the key for exploiting the parallelism of the multi-core processors. Several techniques have been proposed for program multithreading, typically DOALL, DOACROSS, DSWP, PS-DSWP (Prabhu *et al*., 2011).

DOALL parallelization: DOALL parallelization methods of the typical IMT technique are highly efficient because of none the cross-thread communication among iterations. DOALL executes loop iterations in separate threads to prove none inter-iteration dependences. The requirement that the iteration space be divisible prohibits pointer-chasing loops, which are common in general purpose applications, so DOALL parallelization methods usually is used to parallel scientific programs.

DOALL loop algorithms are divided into two categories, shared memory with uniform memory and non-uniform memory access (NUMA). DOALL algorithms that shared memory with uniform memory DOALL have static and dynamic methods. Block Scheduling (BS), static chunking and cyclic scheduling is typical static DOALL loop algorithm. The advantage of static DOALL loop algorithm is lower runtime scheduling overhead while the disadvantage is unbalanced distribution of load among processors. Dynamic loop scheduling algorithms for DOALL loops are self-scheduling (SS),fixed-size chunking (FS), guided self-scheduling (GSS),factoring, trapezoid self-scheduling (TSS) and etc. The advantage of dynamic scheduling is better load balance because of determines the division of iterations among processors at runtime while disadvantage is high overhead. DOALL algorithms

Table 1: Typical DOALL algorithms comparison

Algorithm	Shared Memory	Scheduling	Content
Block scheduling	Uniform memory	Static	Assign iterations 1-N/P. to the first processor, iteration N/P+1-2*N/P to the second processor
Cyclic scheduling	Uniform memory	Static	Allocate iterations i, i+P, i+2P, ..., to processor i (1, i,P)
Fixed-size chunking	Uniform memory	Dynamic	Scheduling steps is P. Chunk size is N/P
Guided self-scheduling	Uniform memory	Dynamic	Scheduling steps is P* ln[N/P]. Chunk size is R/P
Factoring	Uniform memory	Dynamic	Scheduling steps is P*0.44* ln N/P, Chunk size is R/[2P]
Affinity scheduling	NUMA	/	The iterations of a loop are divided into chunks of size N/P iterations and each chunk is statically assigned to a different processor
Dynamic partitioned affinity scheduling	NUMA	/	Similar to AFS, except that it balances the load by readjusting the sizes of the allocated partitions on subsequent executions of a loop
Locality-based dynamic scheduling algorithm	NUMA	/	Adapt to any data partitioning methods, including cyclic or block-cyclic? it computes the chunk size as R/(2P)

Table 2: Typical DOACROSS algorithms comparison

Algorithm	Model	Advantages	Disadvantages
Cyclic	Regular	Exploit the parallelism.	Higher inter-processor communication cost, lower hardware utilization
PSS	Regular/ irregular	Eliminate processor idle cycles and balance the load / have fully parallel inspector and executor phases	Difficult to arrive at an optimal chunk size/ require many memory accesses; it does not treat input dependences differently
ZYRP	Irregular	Can handle all types of dependences, tightly counected the inspector and the executor, simple algorithm	Un-reused inspector of the same loop; the execution of iterations with dependences caunot be overlapped even partially
PRPS	Regular/ irregular	Use the inspector to simply detect whether or not the loop is fully parallel	Cannot exploit partially parallel loops
Chen's CYT	Regular/ irregular	Can reuse the inspector results across loop invocations. It allows the partial overlap of dependent iterations.	Have unnecessary overhead; this algorithm does not treat input dependences differently

of NUMA systems have affinity scheduling (AFS), dynamic partitioned affinity scheduling, wrapped partitioned affinity scheduling (WPAS), locality-based dynamic scheduling algorithm (LDS) (Hurson *et al*., 1997). Some typical DOALL techniques are compared from shared memory, scheduling and content in Table 1.

DOACROSS parallelism: Because data dependences and control dependences among the iterations of the loop are widespread in many applications, so DOACROSS parallelism is aim for these applications. DOACROSS parallelization executes adjacent iterations in coarse-grain parallelism and fine-grained reference-level parallelism by the concurrent execution of parts of each loop iterations across multiple cores. At first one thread executes the loop's critical path on the core. And then execution of the loop's next iteration begins on the next core while the current iterations are being executed. The advantages of DOACROSS parallelization are reducing overhead and handling more general class of loop. The disadvantage of DOACROSS is lower parallel efficiency (Oplinger *et al*., 1997).

Cyclic Scheduling (Cyclic), Staggered Distribution Scheme (SD), Cyclic Staggered Distribution (CSD), Pre-synchronized scheduling (PSS), Zhu-Yew's Runtime Parallelization Scheme (ZYRPS) , Krothapalli's Runtime Paralleliztion Scheme (KRPS), Saltzs Runtime Parallelization Schem (SRPS), Pauchwerger's Runtime Parallelization Schem(PRPS) and Chen's CYTare common DOACROSS Loop Scheduling (Thulasiraman *et al*., 1995). Some typical DOAROSS techniques are compared in Table 2.

DSWP techniques: DSWP techniques can extract threads from general purpose applications, because it uses the fine-grained pipeline parallelism to handle complicated control dependence and memory dependence. DSWP parallelizes a loop by partitioning the body of the loop into a sequence of pipeline stages. Each stage is then executed by a separate thread. The threads communicate either through special hardware structures or through memory. The DSWP algorithm has three main steps. First, the Program Dependence Graph (PDG) which contains all register, memory and control dependences in the loop is constructed. Second, Strongly Connected Components (SCC) are established by founding all dependence recurrences in the PDG. Third, DAGSCC is formed by allocating each SCC to a thread while ensuring that no cyclic dependences are formed between the threads (Vachharajani *et al*., 2007).

The DSWP techniques were first proposed to effective explore parallel of complicated loops (Rangan *et al*., 2004). The DSWP techniques which included a generic algorithm were proposed to automatically apply for the general-purpose applications. The AutoDSWP techniques can identify more loops for the universal applications and ensure that cross-thread iterations does not exist (Rangan, 2007). Because the PS-DSWP techniques integrated the applicability of

Table 3: Typical DSWP algorithms comparison

Algorithm	Compiler	Key insight
Rangan DSWP	/	It identifies two paths of critical path and off-critical path handle parallelized recursive data structure codes
Ottoni DSWP	IMPACT	It uses general algorithm to depart from the notion of identifying critical and off-critical paths of the regions. It achieves pipelined parallelism by that the loop critical path dependence need not even once be routed from core to core. It can improve the thread balance to achieving optimal speedup
AutoDSWP	VELOCITY	It schedules the critical and off-critical path thread pipelines to linear and nonlinear type. It can identify more generic program recurrences and achieves acyclic dependence flow among threads by ensuring that no single recurrence crosses thread boundaries
PS-DSWP	VELOCITY	It's algorithm based on Ottoni DSWP. After isolating the recurrences in their own stages in DSWP, it adopts DOALL techniques to explore the portions of loop

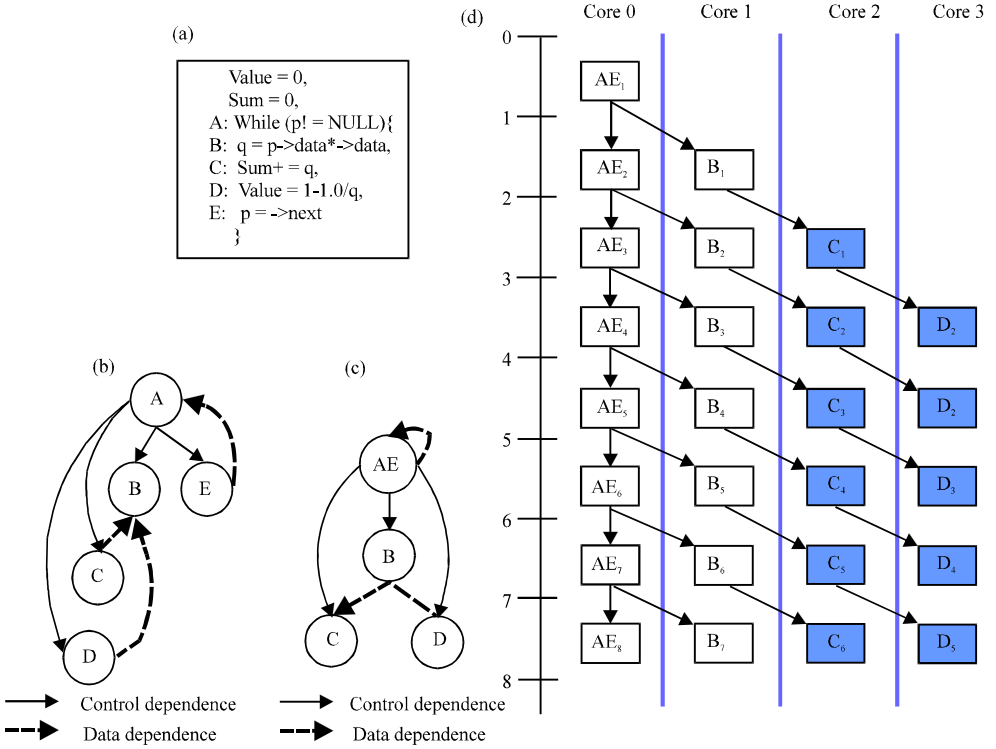

Fig. 3(a-d): DSWP Example, (a) Code, (b) PDG, (c) DAG_{SCC} and (d) Execution of code in figure (a) with DSWP

DSWP with the scalability of DOALL (Raman, 2009), so PS-DSWP techniques implementation are two steps. First, it adopts DSWP to divided the threads, these threads are referred as DSWPed thread. Second using DOALL techniques parallel executed DSWPed thread (De Lima Ottoni, 2008). The execution of DSWP/PS-DSWP techniques is respectively described in Fig. 3 and 4 some typical DSWP techniques are compared in Table 3.

SPECULATIVE THREAD PARTITIONING TECHNIQUES BASED ON MULTI-CORE

The automatic parallelization techniques of DOACROSS, DSWP and PS-DSWP can explore the parallel from sequential application, but much dependence is difficult pre-detection by the non-speculative transformation. So many speculative techniques, such as thread level speculation (TLS), Speculation DSWP (SpecDSWP), Speculative PS-DSWP (SpecPS-DSWP) and Interprocedural SpecPS-DSWP (iSpecPS-DSWP) are proposed to break dependences to increase the parallelism.

All speculative loop parallelization techniques use speculation to break dependences among threads. To ensure the application correct execution, speculative threads maintain speculative state separately from non-speculative state. This allows them to rollback to a known good state when occur mis-speculation (Li and Zhang, 2012).

Speculative techniques based on DOACROSS- TLS: TLS techniques are extended the basic DOACROSS concept with speculation (Hall *et al*., 2005) to aim at reducing inter-thread communication. TLS techniques allow cross-thread dependences and adopt synchrony to maintain correct execution dependences among threads

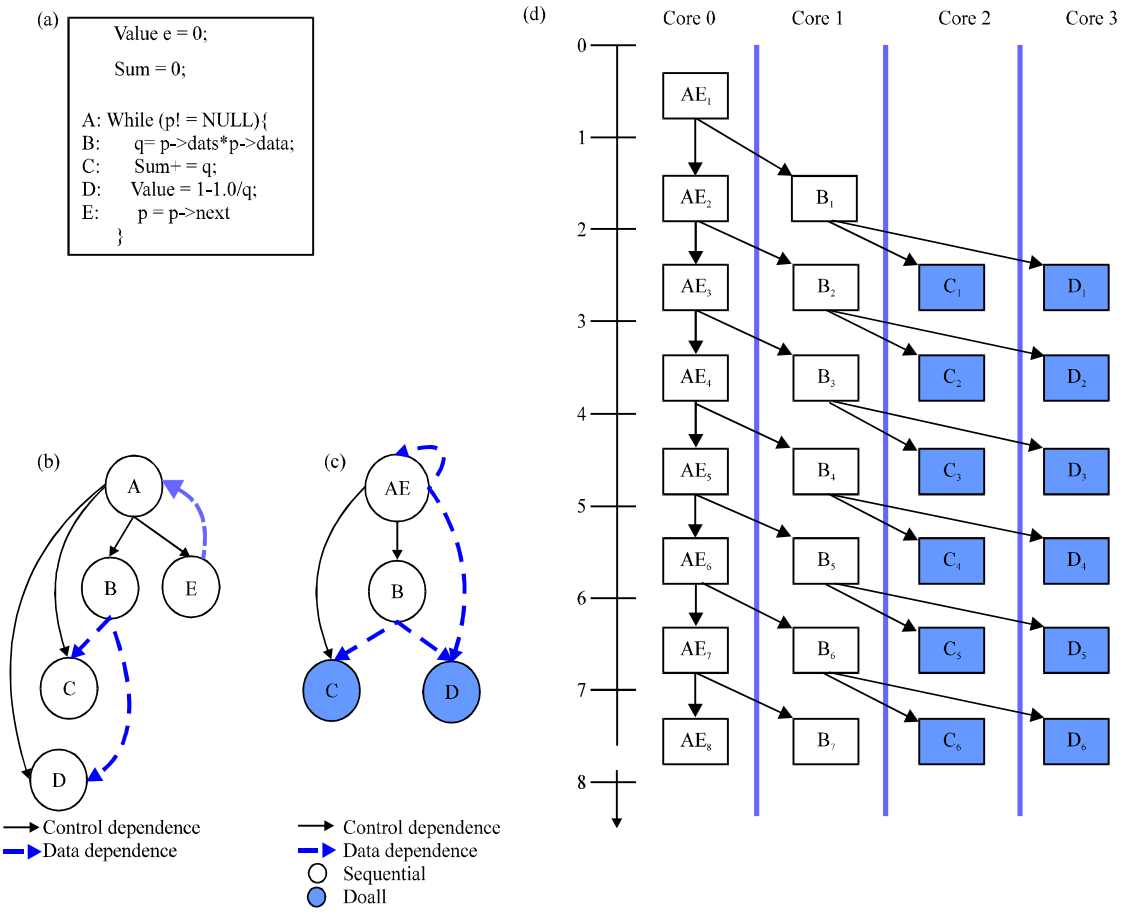

Fig. 4(a-d): PS-DSWP example, (a) Code, (b) PDG, (c), DAG_{SCC} and (d) Execution of code in figure (a) with PS-DSV

(Nemanich, 2009; Bridges, 2008). Like DOACROSS techniques, TLS techniques adopt a separate thread to implement each iteration and parallel execute multiple iterations of a loop. Unlike DOACROSS, TLS techniques execute the oldest thread in sequential application order with non-speculation and execute other threads with speculation (Rauchwerger and Padua, 1999; Zhai, 2005). During the program execution, the sequence of instructions can be split into a set of ordered threads to guarantee the threads committing in original order. Using hardware or software to track all related threads, TLS can ensure that sequential programs are correct of parallel execution on multi-core processors. If the conflicts of the parallel execution is found during the program execution, one of the conflict threads is revocation and then implement the compensation operation and re-execute thread (Olukotun *et al*., 2009). The inter-thread data dependence, inter-thread control-flow mis-prediction and inter-thread load imbalance are the key factors of runtime overhead in TLS techniques (Wang *et al*., 2008; Tian, 2010; Venkatesan, 2009). According to architectures, TLS techniques are divided into tightly coupled TLS architectures, chip multiprocessor TLS, multithreaded processor TLS, shared-memory multiprocessor TLS and software-only TLS (Warg, 2006; Packirisamy, 2009).Trace processor, the superthread architecture, Multiscalar, speculative multithreaded (SM) processor and point are typical tightly coupled TLS architectures. Hydra, STAMPede and POSH compiler are typical chip multiprocessor TLS. Dynamic multithreading (DMT) and implicily multithreaded processors (IMT) are multithreaded processor TLS. Knight and the super threaded architecture are typical shared-memory multiprocessor TLS. Typically software-only TLS compilers are Lazy Privatizing Doall test (LPRD), Behavior-oriented Parallelization (BOP) and Safe Future (Rotenberg *et al*., 1997; Akkary and Driscoll, 1998; Sohi, 2001; Oancea and Mycroft, 2008; Gao *et al*., 2010, Zhong *et al*., 2007; Welc *et al*., 2005). The execution of TLS techniques is described in Fig. 5 and some typical TLS techniques are compared from focus, software and hardware in Table 4.

Table 4: Comparison of typical TLS techniques

Systems	Year	Focus	Software	Hardware
Trace processor	1997	Instruction trace	/	Processing units and local registers
SM	1998	Loops	/	Thread units
Multiscalar	1995	Tasks	Identify and compile tasks	Address resolution buffer and temporal or sequenced cache/buffers
point	2005	Generic	Compiler infrastructure	Speculative multi-threading processor
Hydra	1996	Loops and module-level parallelism	Profile	Speculative execution of threads
STAMPede	2000	Loops	Synchronization	Speculative cache coherence
POSH		Loops and module-level threads	Profile	Shared memory processor
DMT	1998	Loop and module continuations	/	Multi-threaded hardware
IMT	2003	Generic	Identify and compile tasks	Multithreaded hardware with speculative execution of threads
Super threaded	1998	Loops	No data speculation in threads	Multiprocessor with instruction set extensions
LPRD	1995	Loops	Run-time support mark and test phase	/
Safe Future	2005	Loops	Run-time, object copy, byte rewrite	/

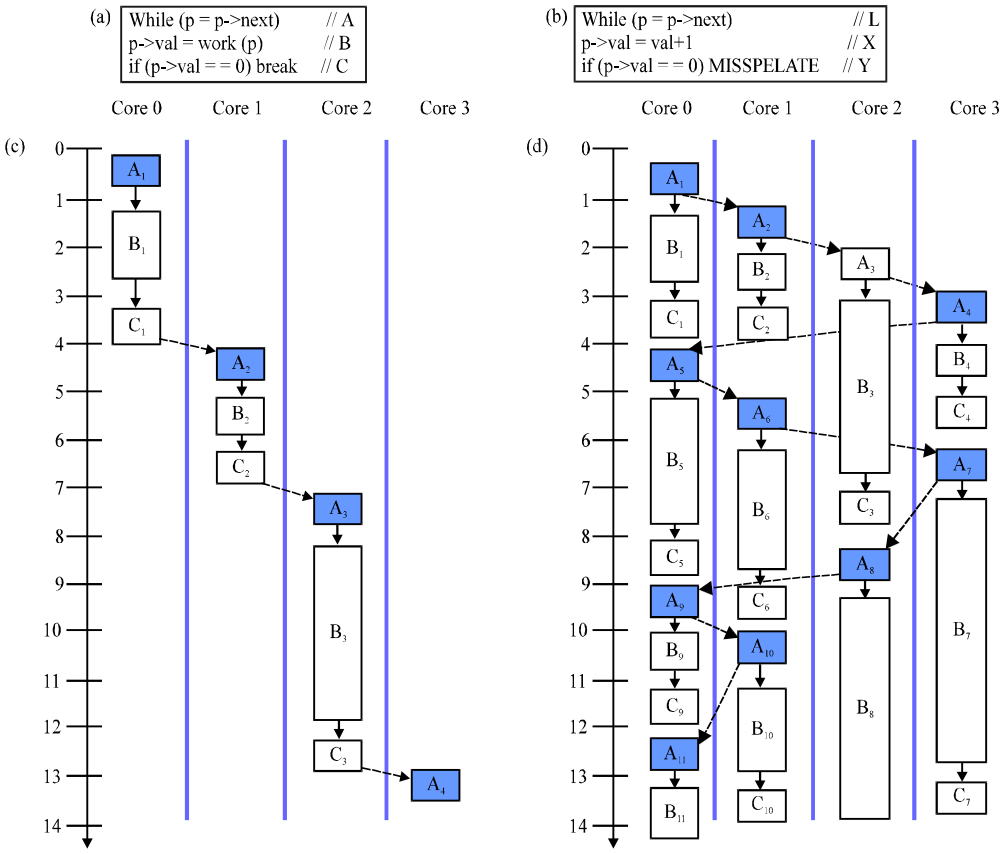

Fig. 5(a-d): TLS example, (a) Sequential code, (b) Speculated sequential code, (c) Execution of code in figure a, (d) Execution of code in figure b

Speculative techniques based on DSWP: Speculation DSWP (SpecDSWP) (Shi, 2007; Vachharajani, 2008) SpecPS-DSWP and iSpecPS-DSWP are typical techniques which add speculative techniques based on DSWP technique.

SpecDSWP extended on DSWP to focus on removing dependences that form large pipeline stages. As in DSWP, SpecDSWP execute stages to achieve pipelined parallelism; Unlike DSWP, SpecDSWP speculative executes each iteration to allow intra-iteration speculation at frequently silent store. SpecDSWP techniques generally checkpoint memory at the beginning of each iteration and executes iteration on multiple threads or multiple processors. SpecDSWP allows multiple threads to execute inside the same version at the same time by a version memory system. If Mis-speculation occurs, SpecDSWP techniques use two recovery mechanisms. One is that worker threads run a non-speculative multithread version of the loop. The other is that commit thread run the original single thread loop. The main

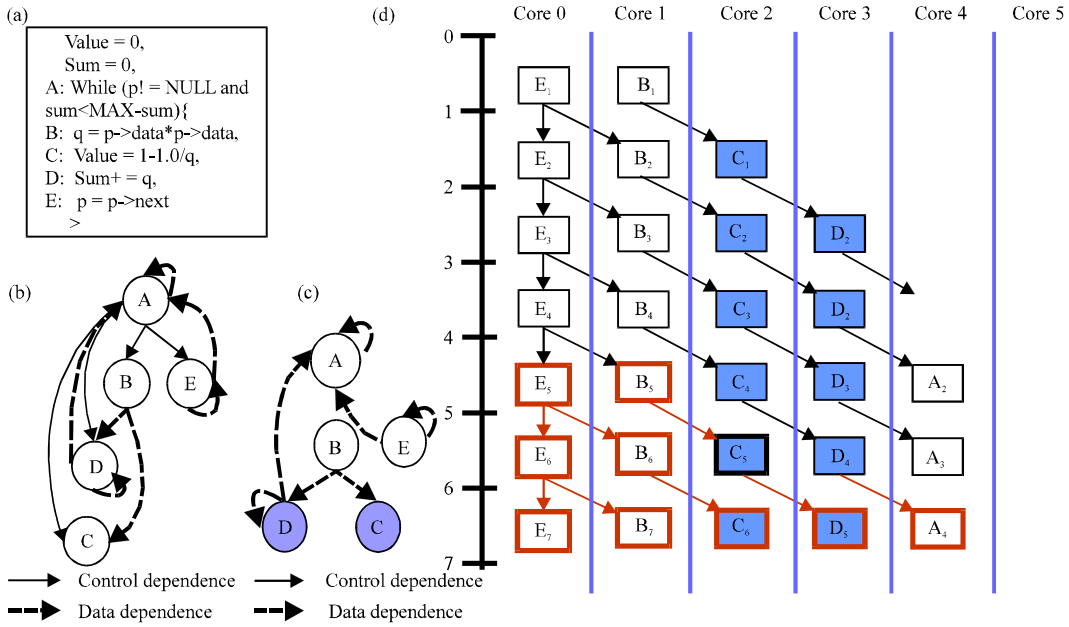

Fig. 6(a-d): SpecDSWP example, (a) Code, (b) PDG, (c) DAG_{SCC} and (d) Execution of code with figure (a) with DSWP

advantage of SpecDSWP is its latency tolerant property realized by decoupling inter-process communication delay from on the critical path to off the critical path.

The SepcPS-DSWP's performance of are limited by the static analyses. So that, SpecPS-DSWP techniques add speculation to the PS-DSWP algorithms to extract large amount of parallelism. SpecPS-DSWP can use speculation to remove inter-iteration dependences to prevent the extraction of data-level parallelism. Reifying the speculation and then applying the PS-DSWP algorithm to the resulting code assigns the SpecPS-DSWP thread. After generating PS-DSWP multi-threaded code, checkpoint and recovery code are inserted into each thread and the Commit Thread. SpecPS-DSWP relies upon a version memory system with hierarchical memory versions. The iSpecPS-DSWP technique is a novel parallelization framework to optimize larger loops than existing parallelization techniques.

The iSpecPS-DSWP technique is formed that add inter-procedural analysis and optimization scope to the existing PMT techniques, including DSWP, PS-DSWP, Spec-DWSP and SpecPS-DWSP (Bridges, 2008). iSpecPS-DSWP builds interprocedural PDG(iPDG) which is derivative of the system dependence graph(SDG). iPDG contains all local dependence and any reachable procedures. Deciding the loop-carriedness of each edge in the iPDG will occur register dependence, parameter dependence, control dependence, call dependence and memory dependence. If register dependence, control dependence and memory dependence are in the DSWPed loop, these dependences are treated as SecpPS-DSWP. Otherwise, they are marked as intra-iteration (INTRA). Parameter dependence and call dependence are always marked as INTRA because they can not involve the loop back edge. Parameter dependence is handled similar to register dependences. Call dependences are treated in the same as the control dependences. The execution of SpecDSWP/SpecPS-DSWP techniques is respectively described in Fig. 6 and 7.

SpecDSWP, SpecPS-DSWP and iSpecPS-DSWP are typical speculative DSWP techniques. They are compared from based techniques, recovery mechanism, support speculative types and complement steps. In addition to control speculation, SpecDSWP, SpecPS-DSWP and iSpecPS-DSWP support other speculative types, such as biased branch speculation, infrequent basic block speculation, memory value speculation and silent store speculation. Biased branches speculation can break the control dependences between the branch and other instructions and break data dependences by altering the control flow of the program. The compiler inserts a new basic block on each control flow edge from the branch to each speculated target to realize biased branches speculation. Infrequent block speculation can be mapped to a set of biased branch speculation to break all incoming control flow edges in blocks. To enable silent store speculation, the compiler inserts a load of the same memory and a branch to compare the loaded value with the one about to be stored. The memory dependences are removed by silent store speculation when silent stores write to a location but they do not change the location's value. To realize memory value speculation, the compiler

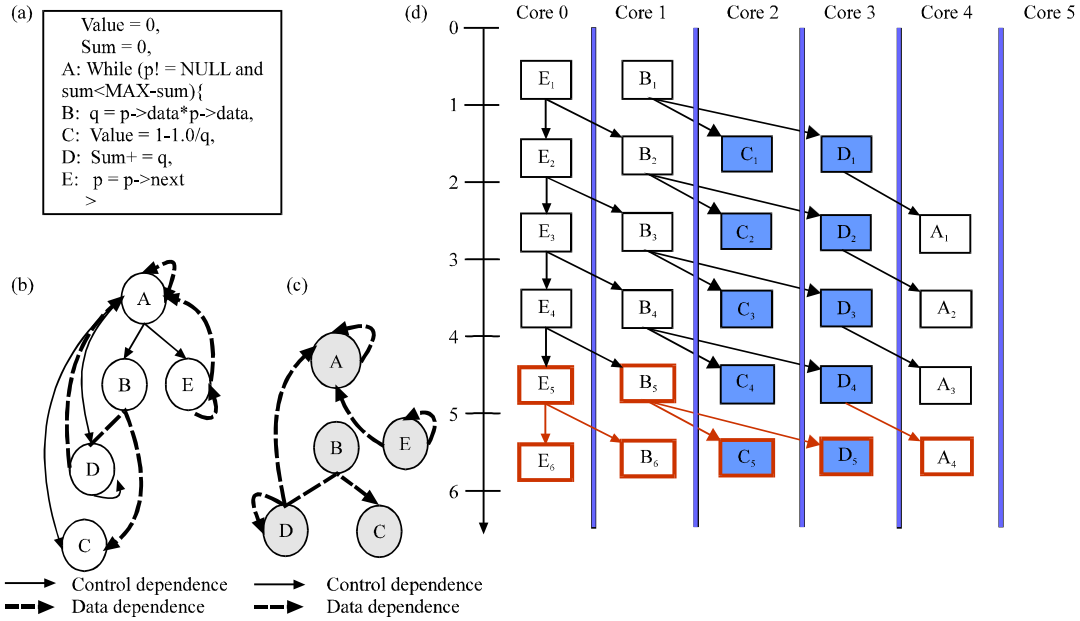

Fig.7(a-d): SpecPS-DSWP example, (a) Code, (b) PDG, (c) DAG_{SCC} and (d) Execution of code in figure a with DSWP

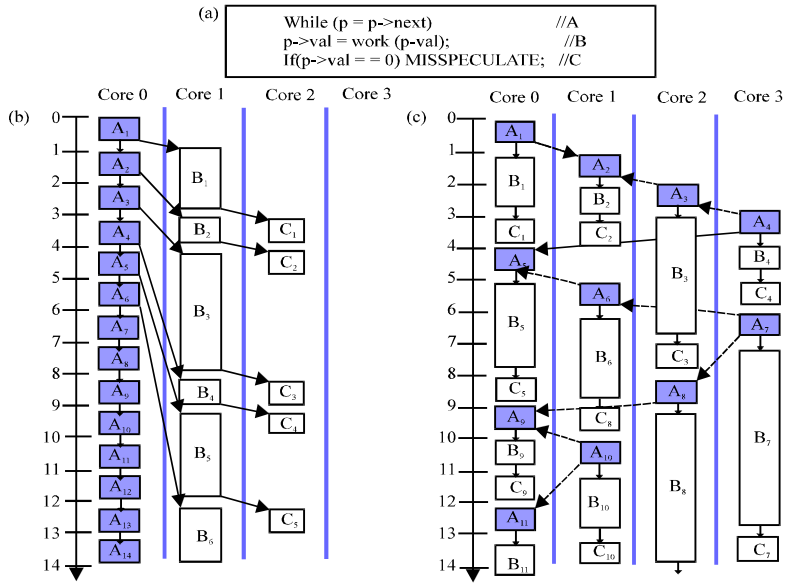

Fig. 8(a-c): TLS and Spec-DSWP schedule, (a) Code, (b) Execution of code in figure a with SpecDSWP and (c) Execution of code in figure a with TLS

compares two value and use the same way of biased branch speculation to treat the differ value. The loop-carried invariant speculation, parameter dependence speculation and call dependence speculation are processed by the same methodology as memory value speculation. Some typical Speculative techniques based on DSWP are compared in Table 5.

Although either TLS or SpecDSWP technique could extract parallelism from loops, the TLS and SpecDSWP technique has their advantages. For TLS, it more suitable for general program and its speed not limited by the size of largest pipeline stage. For SpecDSWP, it can better handle synchronizing dependences among iterations and has not communication latency that caused by the critical path for the synchronized portion (Bridges, 2008). The difference of TLS and SpecDSWP techniques is described in Table 6 and using Fig. 8 describe the execution of the both techniques.

Table 5: Comparison speculative DSWP algorithms

Algorithm	Based Techniques	Recovery mechanism	Support speculative types	Complement Steps
Spec-DSWP	DSWP	Version memory	1.Biased branch speculation	1.Build PDG
			2.Infrequent basic block speculation	2.Select speculation
			3.Silent store speculation	3. Build speculative PDG
			4.Memory value speculation	4.Form SCC
				5.Detect misspeculation
				6.Determine need speculation
SpecPS-DSWP	PS-DSWP	Hierarchical memory versions		1.Build PDG
			1.Biased branch speculation	2.Select speculation
			2.Infrequent block speculation	3.Build speculative PDG
			3.Silent store speculation	4.Form and assign SCC
			4.Memory value speculation	5.Determine need speculation
			5.Loop-carried invariant speculation	6.Rebulid speculative PDG
				7.Form and assign SCC
				8.Determine memory versions
				9.Determine synchronization
iSpecPS-DSWP	DSWP	Hierarchical memory versions	1.Biased branch speculation	1.Build iPDG
	PS-DSWP		2.Infrequent block speculation	2.Select speculation
	SpecPS-DSWP		3.Silent store speculation	3.Build speculative iPDG
			4.Memory value speculation	4.Form and assign SCC
			5.Loop-carried invariant speculation	5.Determine need speculation
			6. Parameter dependence speculation	6.Rebulid speculative iPDG
			7. Call dependence speculation	7.Form and assign SCC
				8.Determine memory versions
				9.Determine synchronization

Table 6: Comparison TLS and SpecDSWP algorithms

Algorithm	TLS	SpecDSWP
Focus on	Remove inter-iteration	Reduce inter-thread communication
Iteration	Stay local on a single thread processor	Stay on multiple processors
Limited in speed	The size of the longest cross-iteration dependence cycle communication latency between processor cores	The size of largest pipeline stage
Submitted order	Older iteration's commit before younger iterations when conflict occurs, younger iteration roll back	When conflict cocurs, each thread is restored to its recovery code Restore threads' state and re-executing iteration non-speculatively
Implement model	Iteration executes in a single thread speculative execute old thread, non-speculative execute other	Execute each iteration in a different thread, forming pipeline only in a single direction All iterations executes speculatively

THREAD-LEVEL SPECULATION TECHNIQUES ON TRANSACTIONAL MEMORY

Transactional Memory (TM): Transactional Memory (TM) systems adopt implicit locks to exploit more parallelism and check the read/write conflict before committing a transaction. When TM systems occurs the conflicts, TM systems aborts the transaction and rolls back and then processes the transaction again until they have no conflict. TM systems can be classified into hardware TM (HTM), software TM (STM) and hybrid TM (HyTM) according to the different implementations (Nasir, 2009). HTM system implement conflict detection and consistency among concurrent transactions by hardware cache. The advantages of HTM are the higher speed, easier programming and lower overhead. The disadvantage of HTM is that it must be redesigned the caches and the coherence protocol (Grahn, 2010). Some HTM designs were proposed, such as, transactional coherence and consistency (TCC) (Hammond *et al*., 2004), Bulk (Ceze *et al*., 2006), unbounded TM (UTM), large TM (LTM) (Ananian *et al*., 2005) and token TM (Bobba, 2010) systems. STM systems can be designed to work on existing systems without adding extra special hardware. STM systems adopt software to deal conflict detection, implement version management, guarantee the synchronization and etc. The advantages of STM are easier modification and extending. But STM execution has higher overhead and lower performance than HTM. Some STM designs were proposed, such as, Rochester software transactional memory (RSTM) (Marathe *et al*., 2006), word based software transactional memory (WSTM) (Harris and Fraser, 2003), Intel's McRT-STM (Guerraoui *et al*., 2007), Dynamic Software Transactional Memory (DSTM) (Herlihy *et al*., 2003) and transactional locking II (TL2) (Dice *et al*., 2006). HyTM aims at using the advantages of both HTM and STM. HyTM adopt software to execute some larger transaction while HyTM adopt hardware to execute part functions of systems and some small transaction. Signature-accelerated transactional memory (SigTM) (Minh *et al*., 2007), Virtualized TM (VTM) (Rajwar *et al*., 2005), log-based

transactional memory (LogTM) (Damron *et al*., 2006), logTM signature edition (LogTM-SE) and non-blocking zero-indirection transactional memory (NZTM) (Al Tabba, 2011) are some typical HyTM (Li *et al*., 2010).

Extended TM to support TLS: There are many differences semantic between TLS and TM but both of them require similar underlying support, such as dependency violation detection, result buffering, check pointing and replay. TM systems lack thread spawning mechanism, context passing mechanism and sequential ordering mechanism compared to TLS. Therefore, combined the TLS and TM can more effectively improve the serial program running on the multi-core processors (Zhong *et al*., 2007; Zhang *et al*., 2012b). Typical systems which support to TLS and TM techniques are TCC, STMlite, Helper Transaction and PTT. They all adopt compiler to support the context passing mechanism. TCC adopts hardware to support sequential ordering commit and adopts manual and hardware to implement thread spawning. Helper Transactions (Yoo and Lee, 2008) is an in-depth design that can effectively exploit out-of-order procedure speculation. Helper Transactions uses binary tree to describe the sequential ordering mechanism, while it uses compiler to spawn thread and context passing mechanism. STMlite (Mehrara *et al*., 2009) support TLS technology based on light-weight STM model. STMlite adopts hash read and write sets to spawn thread and employs transaction commit manager to guarantee sequential ordering commit. Profile of a guided TLS and TM system (PTT) system (Wang, 2010) which expands the token in LogTM adopts determination priority, modification cache coherence protocol and token mechanism to support sequential committing. PTT modifies cache coherence protocol and hardware to spawn thread (Li *et al*., 2012).

CONCLUSION

In this study, we introduced three parallel types of thread level parallelism, such as IMT, CMT, PMT and described their execution model. We compared typical DOALL, DOACROSS, DSWP and PS-DSWP techniques. These techniques can explore the parallel from sequential application but much dependence is not easily predictable or manifests them infrequently by the non-speculative transformation. So many speculative techniques, such as TLS, SpecDSWP, SpecPS-DSWP and iSpecPS-DSWP, are proposed to further improve parallelism. We introduced these speculative parallel techniques and described their execution model. SpecDSWP, SpecPS-DSWP and iSpecPS-DSWP are compared from supporting speculation types, memory version and implement steps. We compared TLS and SpecDSWP techniques from iteration, limited in speed, submitted order and implement model. While TLS more suitable for general program, SpecDSWP can better handle synchronizing dependences among iterations. Some opportunities for future research in parallel sequential program are novel thread partitioning algorithms, inter-procedural extensions, speculative optimization and etc. At last, some extended TM to support TLS systems are analyzed from thread spawning mechanism, context passing mechanism and sequential ordering.

ACKNOWLEDGMENTS

This work is supported by the 2007 Nationality 863 Project "Software Product Line based on Workflow" (No. 2007AA010305), the 2013 Shaanxi Provincial Department of Education project "Research of technology and system of thread level speculation based on transactional memory"(2013JK1197), the State Key Laboratory for Manufacturing Systems Engineering Open Project (No. sklms2011011). The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

Akkary, H. and M.A. Driscoll, 1998. A dynamic multithreading processor. Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture, November 30-December 2, 1998, Dallas, TX., USA., pp: 226-236.

Al Tabba, F.A.W., 2011. A study of hybrid transactional memory. Doctoral Dissertation, The University of Auckland, Auckland, New Zealand.

Ananian, C.S., K. Asanovic, B.C. Kuszmaul, C.E. Leiserson and S. Lie, 2005. Unbounded transactional memory. Proceedings of the 11th International Symposium on High-Performance Computer Architecture, February 12-16, 2005, San Francisco, CA., USA., pp: 316-327.

Bobba, J., 2010. Hardware support for efficient transactional and supervised memory systems. Ph.D. Thesis, University of Wisconsin-Madison, USA.

Bridges, M.J., 2008. The velocity compiler: Extracting efficient multicore execution from legacy sequential codes. Ph.D. Thesis, Princeton University, USA.

Ceze, L., J. Tuck, J. Torrellas and C. Cascaval, 2006. Bulk disambiguation of speculative threads in multiprocessors. ACM SIGARCH Comput. Archit. News, 34: 227-238.

Damron, P., A. Fedorova, Y. Lev, V. Luchangco, M. Moir and D. Nussbaum, 2006. Hybrid transactional memory. ACM SIGARCH Comput. Archit. News, 34: 336-346.

Davies, J.R.B., 1981. Parallel loop constructs for multiprocessors. M.S. Thesis, University of Illinois, Urbana, IL., USA.

De Lima Ottoni, G., 2008. Global instruction scheduling for multi-threaded architectures. Ph.D. Thesis, The Faculty of Princeton University, USA.

Dice, D., O. Shalev and N. Shavit, 2006. Transactional locking II. Proceedings of the 20th International Symposium on Distributed Computing, September 18-20, 2006, Stockholm, Sweden, pp: 194-208.

Ferrante, J., K.J. Ottenstein and J.D. Warren, 1987. The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst., 9: 319-349.

Gao, L., J.L. Xue and T.F. Ngai, 2010. Loop recreation for thread-level speculation on multicore processors. Software: Pract. Experience, 40: 45-72.

Grahn, H.J., 2010. Transactional memory. J. Parallel Distrib. Comput., 70: 993-1008.

Guerraoui, R., M. Kapalka and J. Vitek, 2007. STMBench7: A benchmark for software transactional memory. ACM SIGOPS Oper. Syst. Rev., 41: 315-324.

Hall, M.W., S.P. Amarasinghe, B.R. Murphy, S.W. Liao and M.S. Lam, 2005. Interprocedural parallelization analysis in SUIF. ACM Trans. Program. Lang. Syst., 27: 662-731.

Hammond, L., V. Wong, M. Chen, B.D. Carlstrom and J.D. Davis *et al*., 2004. Transactional memory coherence and consistency. ACM SIGARCH Comput. Archit. News, 32: 102-113.

Harris, T. and K. Fraser, 2003. Language support for lightweight transactions. ACM SIGPLAN Not., 38: 388-402.

Herlihy, M., V. Luchangco, M. Moir and W.N. Scherer, 2003. Software transactional memory for dynamic-sized data structures. Proceedings of the 22th Annual Symposium on Principles of Distributed Computing, July 13-16, 2003, Boston, Massachusetts, USA., pp: 92-101.

Hurson, A.R., J.T. Lim, K.K. Kavi and B. Lee, 1997. Parallelization of DOALL and DOACROSS loops: A survey. Adv. Comput., 45: 53-103.

Li, X. and J. Zhang, 2012. Thread level speculation based on transactional memory. Inform. Int. Interdisciplinary J., 15: 1475-1755.

Li, X., F. Li and C.H. Wang, 2012. Research of parallel processing technology based on multi-core. Applied Mech. Mater., 182-183: 639-643.

Li, X., J. Zhang and J.H. Li, 2010. Hardware/hybrid transactional memory. Proceedings of the International Conference on Computer, Mechatronics, Control and Electronic Engineering, Volume 6, August 24-26, 2010, Changchun, China, pp: 68-71.

Marathe, V.J., M.F. Spear and C. Heriot, 2006. Lowering the overhead of software transactional memory. Technical Report No. TR 893, Computer Science Department, University of Rochester, USA., pp: 1-11.

Mehrara, M., J. Hao, P.C. Hsu and S. Mahlke, 2009. Parallelizing sequential applications on commodity hardware using a low-cost software transactional memory. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, June 15-21, 2009, Dublin, Ireland, pp: 166-176.

Minh, C.C., M. Trautmann, J.W. Chung, A. McDonald and N. Bronson *et al*., 2007. An effective hybrid transactional memory system with strong isolation guarantees. ACM SIGARCH Comput. Archit. News, 35: 69-80.

Nasir, M., 2009. Software transactional memory techniques principles, design and implementation trade-offs. Master Thesis, School of Computing, Blekinge Institute of Technology, Sweden.

Nemanich, B., 2009. Designing a sequence L compiler for multi-core processors. Ph.D. Thesis, Texas Tech University, USA.

Oancea, C.E. and A. Mycroft, 2008. Software thread-level speculation: An optimistic library implementation. Proceedings of the 1st International Workshop on Multicore Software Engineering, May 10-18, 2008, Leipzig, Germany, pp: 23-32.

Olukotun, K., L. Hammond and J. Laudon, 2009. Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency. Morgan and Claypool Publishers, USA.

Oplinger, J., D. Heine, S.W. Liao, B.A. Nayfeh, M.S. Lam and K. Olukotun, 1997. Software and hardware for exploiting speculative parallelism with a multiprocessor. Technical Report No. CSL-TR-97-715, Stanford University, USA., pp: 1-23.

Packirisamy, V., 2009. Efficient architecture support for thread-level speculation. Ph.D. Thesis, University of Minnesota, USA.

Prabhu, P., S. Ghosh, Y. Zhang, N.P. Johnson and D.I. August, 2011. Commutative set: A language extension for implicit parallel programming. Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, Volume 46, June 4-8, 2011, San Jose, CA., USA., pp: 1-11.

Rajwar, R., M. Herlihy and K. Lai, 2005. Virtualizing transactional memory. Proceedings of the 32nd Annual International Symposium on Computer Architecture, June 4-8, 2005, Madison, Wisconsin USA., pp: 494-505.

Raman, E., 2009. Parallelization techniques with improved dependence handling. Ph.D. Thesis, Princeton University, USA.

Rangan, R., 2007. Pipelined multithreading transformations and support mechanisms. Doctoral Dissertation, Princeton University, Princeton, New Jersey, USA.

Rangan, R., N. Vachharajani, M. Vachharajani and D.I. August, 2004. Decoupled software pipelining with the synchronization array. Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, September 29-October 3, 2004, Antibes Juan-les-Pins, France, pp: 177-188.

Rauchwerger, L. and D.A. Padua, 1999. The LRPD test: Speculative run-time parallelization of loops with privatization and reduction parallelization. IEEE Trans. Parallel Distrib. Syst., 10: 160-180.

Rotenberg, E., Q. Jacobson, Y. Sazeides and J. Smith, 1997. Trace processors. Proceedings 13th Annual IEEE/ACM International Symposium on Microarchitecture, December 1-3, 1997, Research Triangle Park, NC., USA., pp: 138-148.

Shi, J., 2007. Binary analysis and optimization for explicitly decoupled architectures. Doctoral Dissertation, University of Rochester, Rochester, New York, USA.

Sohi G.S., 2001. Speculative multithreaded processors. Computer, 34: 66-73.

Thulasiraman, J., V.P. Krothapalli and M. Giesbrecht, 1995. Run-time paralleliztion of irregular DOACROSS loops. Technical Report No. 95/03, Department of Computer Science, University of Manitoba, Canada.

Tian, C., 2010. Speculative parallelization on multicore processors. Ph.D. Thesis, University of California, Riverside, USA.

Vachharajani, N., R. Rangan, E. Raman, M.J. Bridges, G. Ottoni and D.I. August, 2007. Speculative decoupled software pipelining. Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques, September 15-19, 2007, Brasov, Romania, pp: 49-59.

Vachharajani, N.A., 2008. Intelligent speculation for pipelined multithreading. Ph.D. Thesis, Princeton University, Princeton, New Jersey, USA.

Venkatesan, P., 2009. Exploring efficient architecture design for thread-level speculation-Power and performance perspectives. Ph.D. Thesis, The University of Minnesota, USA.

Wang, Y.B., 2010. Performance optimization on multicore transactional memory architecture supporting speculative parallelization. Ph.D. Thesis, University of Science and Technology of China, China.

Wang, Y.B., H. An, B. Liang, L. Wang and R. Guo, 2008. A dynamic profiling tool set for exploring thread-level speculation parallelism. Proceedings of the International Conference on Computer and Electrical Engineering, December 20-22, 2008, Phuket, Thailand, pp: 256-260.

Warg, F., 2006. Techniques to reduce thread-level speculation overhead. Ph.D. Thesis, Chalmers University of Technology, Goteborg, Sweden.

Welc, A., S. Jagannathan and A. Hosking, 2005. Safe futures for Java. Proceedings of the 20th Annual Conference on Object-Oriented Programming, Systems, Languages and Applications, Volume 40, October 16-20, 2005, San Diego, CA., USA., pp: 439-453.

Yoo, R.M. and H.H.S. Lee, 2008. Helper transactions: Enabling thread-level speculation via a transactional memory system. Proceedings of the Workshop on Parallel Execution of Sequential Programs on Multi-core Architectures in Conjuction with ACM/IEEE International Symposium on Computer Architecture, June 2008, Beijing, China, pp: 63-71.

Zhai, A., 2005. Compiler optimization of value communication for thread-level speculation. Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA., USA.

Zhang J., X.J. Chen, J.H. Li and X. Li, 2012a. Resource reconstruction algorithms for optimal allocation in virtual computing systems. Int. J. Autom. Comput., 9: 142-154.

Zhang, J., X.J. Chen, J.H. Li and X. Li, 2012b. Task mapper and application-aware virtual machine scheduler oriented for parallel computing. J. Zhejiang Univ. Sci. C, 13: 155-177.

Zhong, H., S.A. Lieberman and S.A. Mahlke, 2007. Extending multicore architectures to exploit hybrid parallelism in single-thread applications. Proceedings of the IEEE 13th International Symposium on High Performance Computer Architecture, February 10-14, 2007, Scottsdale, AZ., USA., pp: 25-36.