Helper Threading with SSMT

ECE 1718 Final Presentation

Kurniadi Asrigo
Chris Comis
Hassan Shojania

April 28, 2004

Outline

1. Motivation
2. Problem Definition
3. Tools
4. Initial Benchmark Analysis
5. First Attempt: A Helper Process
6. Second Attempt: A Helper Thread
7. Summary of Results
8. Conclusions
9. References

Motivation

- Memory accesses lead to pipeline stalls that may only partly be remedied by out-of-order execution
- Increased processor clock speed results in increased latency cost for a data cache miss
- On an SMT machine, a helper thread may be invoked to mask the miss latencies of a single-threaded application

Problem Definition

- To isolate and profile delinquent memory access loads in single-threaded benchmark applications
- To implement multiple helper threading techniques on these benchmarks
- To analyze the delinquent load performance effects, as well as the overall performance effects, using the SSMT simulator

SimpleScalar and SSMT

- SimpleScalar out-of-order
  - A Linux-based simulator that simulates Alpha binaries
  - Allows detailed out-of-order execution, cache, prediction
  - Performs speculative execution
  - Includes a complex memory management unit
  - ~3500 lines of code
- SSMT
  - An extension of SimpleScalar out-of-order to handle SMT processing
  - ~8024 lines of code
  - The Ph.D. thesis of Dominik Madon, École Polytechnique Fédérale de Lausanne
  - With bug fixes/updates from University of Maryland

The Alpha Machine

- CITA (Canadian Institute for Theoretical Astrophysics)
  - Cluster of 256 dual Xeon nodes; ranking #70 in top500
  - Compaq "Wildfire" GS320 with 32 alpha processors
  - Quad AlphaServer ES45 with 1 GHz EV68 (21264) processors ✓

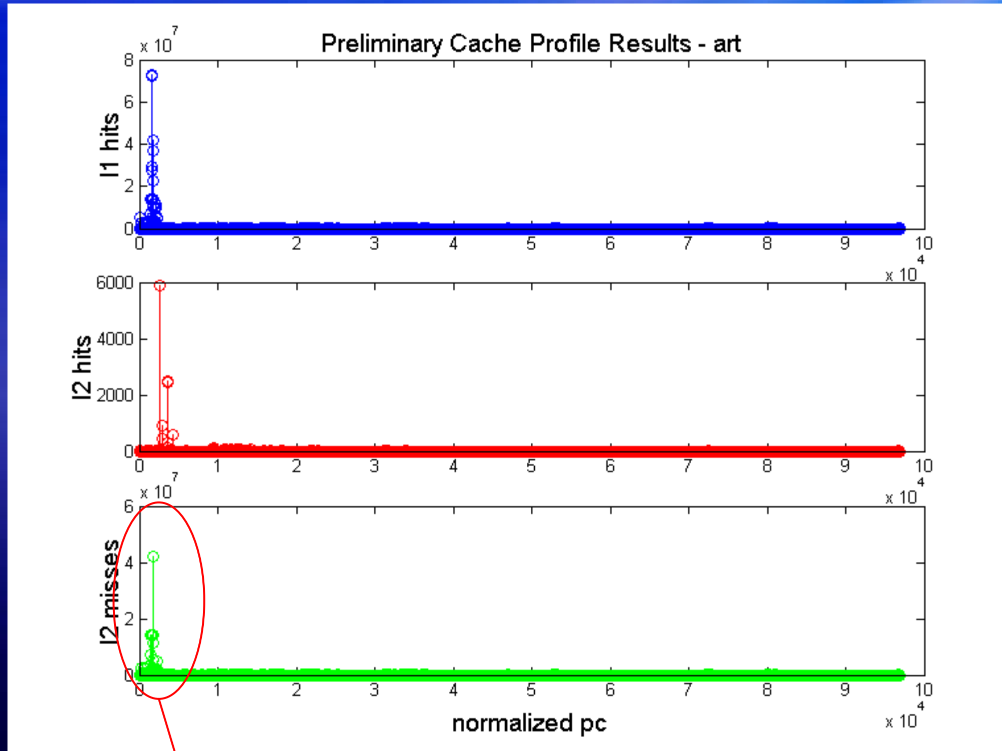
Benchmarks

- Spec2000 – art and mcf
- Spec95 – go
- The main focus on this project was on mcf

Preliminary Benchmark Analysis

- The SSMT simulator was given the following setup (to synchronize with results presented in [1])
  - L1 Cache (non-shared, LRU)
    - Size of 16k
    - Associativity of 4
    - 64 Sets
  - L2 Cache (shared, LRU)
    - Size of 512k
    - Associativity of 8
    - 1024 Sets
- The SSMT simulator was modified to record data memory accesses (L1 hits, L2 hits, L2 misses)

Preliminary Benchmark Analysis - art

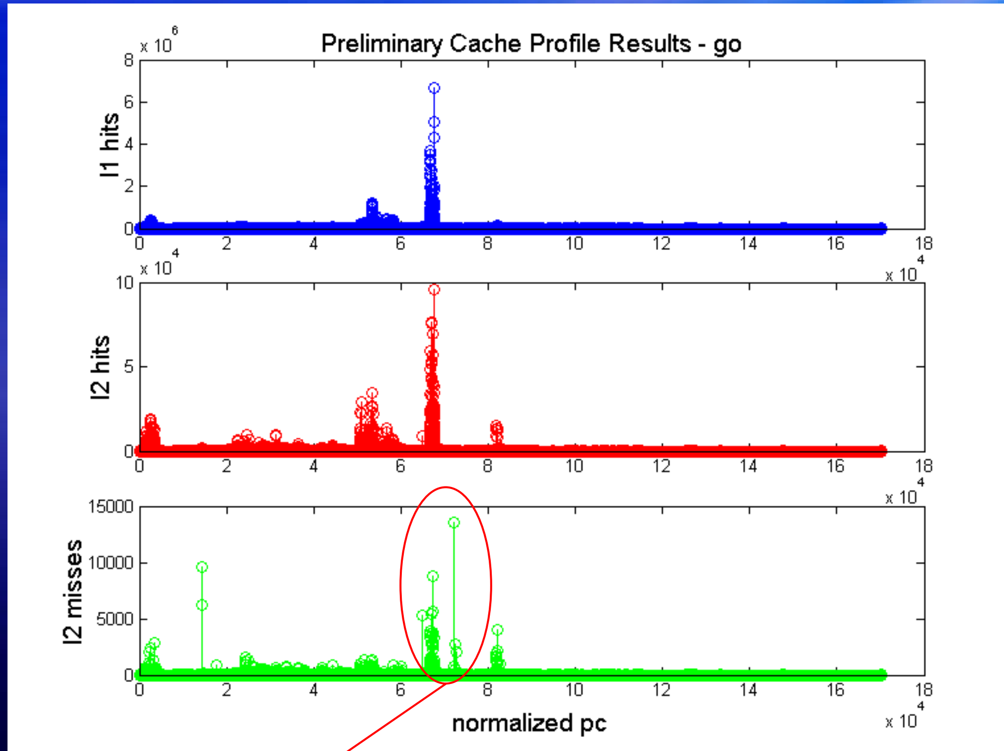


The most delinquent PC!

art results:

- Simulation time: 13 hours
- Delinquent PC: 0x200100F4
- L2 misses: 42090000
- L2 hits: 5896
- L1 hits: 190463

Preliminary Benchmark Analysis - go

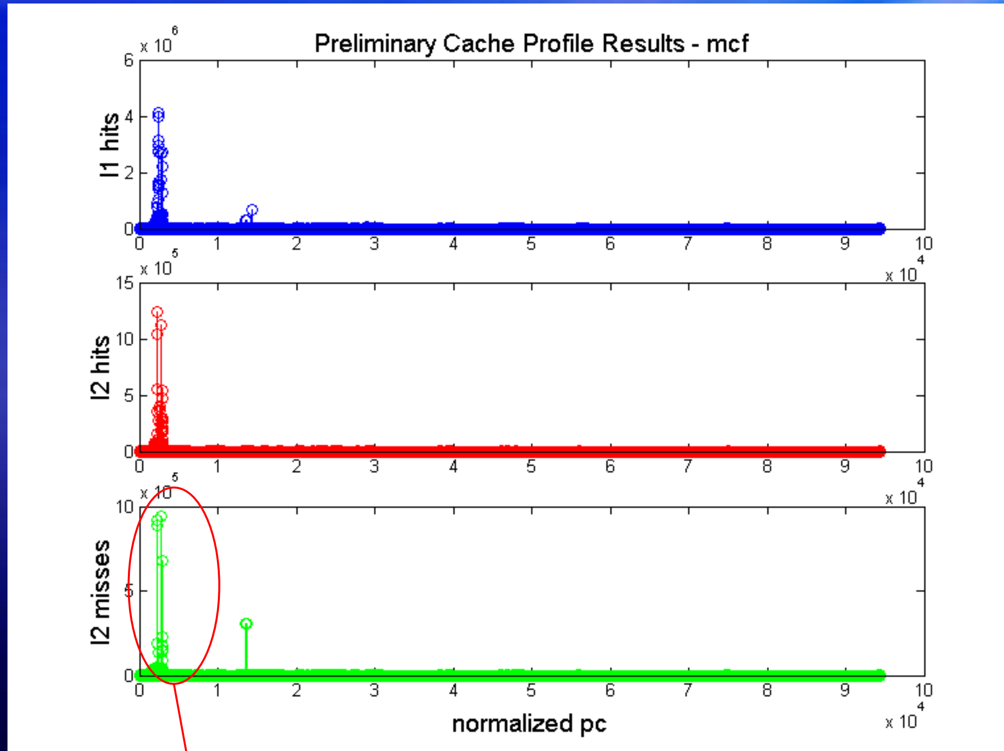


The most delinquent PC!

go results:

- Simulation time: 90 minutes
- Delinquent PC: 0x20055778
- L2 misses: 13594
- L2 hits: 96043
- L1 hits: 1113605

Preliminary Benchmark Analysis - mcf



The most delinquent PC!

mcf results:

- Simulation time: 40 minutes
- Delinquent PC: 0x2000f558
- L2 misses: 942556
- L2 hits: 1117430
- L1 hits: 347944

VTune

VTune(TM) Performance Analyzer - [Source View - [f:\...t2000\181.mcf\run\00000001\pbeampp.c]]

File Edit View Activity Configure Window Help

Activity1 (Sampling)

Tuning Browser

- VTProject4
  - Activity1 (Sampling)
    - Sampling Results [PC
      - Run 1
        - 2nd Level Ca
        - MOB Loads I
        - Clockticks
      - Run 2
        - Split Loads F
      - Run 3
        - 2nd-Level Ca
        - x87 Input As
      - Run 4
        - x87 Output A

Address	Line	Source	2nd Lev	MOB	Clocktick	Split Lo	2nd
0x2ECF	168	perm[next]->a = arc;		1			
	169	perm[next]->cost = red_cost;					
0x2ED7	170	perm[next]->abs_cost = ABS(red_cost);		1	3		
	171	}					
	172	}					
0x2EFA	173	basket_size = next;					
	174	}					
	175						
0x2F00	176	old_group_pos = group_pos;					
	177						
	178	NEXT:					
	179	/* price next group */					
0x2F14	180	arc = arcs + group_pos;					
0x2F1F	181	for(; arc < stop_arcs; arc += nr_group)					
	182	{					
0x2F26	183	if(arc->ident > BASIC)	143	6	75		
	184	{					
0x2F2D	185	red_cost = bea_compute_red_cost(arc);	19	6	17		
0x2F3B	186	if(bea_is_dual_infeasible(arc, red_cost)	12	7	7		
	187	{					
	188	basket_size++;					
0x2F4B	189	perm[basket_size]->a = arc;			1		
	190	perm[basket_size]->cost = red_cost;					
0x2F53	191	perm[basket_size]->abs_cost = ABS(red_co	4	4	4		
	192	}					
	193	}					
	194						
	195	}					
	196						
0x2F7F	197	if(++group_pos == nr_group)					
0x2F84	198	group_pos = 0;					
	199						
0x2F86	200	if(basket_size < B && group_pos != old_group_pos)					
	201	goto NEXT;					

Function Summary				Sampling Results [PCMAHDI] - Mon Apr 12 09:22:08 2004	
Address	Size	Function	Class	2nd Level Cache Read Misses (13)	MOB Loads Replays Retired (13)
-----	-----	--- Selected Range ---	-----	143	6
0x2D80	0xBA	sort_basket		0	0
0x2E40	0x1A1	primal_bea_mpp		179	2

Intel® Tuning Assistant

Top Insights | Workload Insights | Mo Ins | Hotspot Insights | System Info

Hint: Many of the statistics displayed by the Tuning Assistant are heuristic only.

Tuning Analysis for Sampling Results [PCMAHDI] - Mon Apr 12 09:22:08 2004

Top 5 Hotspot Insights

Time-Based Coding Pitfalls

- Blocked Store Forwards: 0.002 sec processor time Seen in primal_bea_mpp (RVA: 0x2e40-0x2fe0, process: mcf.exe, module: mcf.exe)
- Blocked Store Forw 0.0017 sec process Seen in refresh_neighbo (RVA: 0x1330-0x1 process: mcf.exe, mcf.exe)

<< Page: 1 of 8 >>

For Help, press F1

Preliminary Benchmark Analysis

- The art benchmark was eliminated because of the long simulation time
- The go benchmark was eliminated because it did not have a large delinquent load
- The mcf benchmark has a good delinquent load (942556 L2 misses) and runs in reasonable simulation time (40 minutes)
- The mcf benchmark will be analyzed in further detail

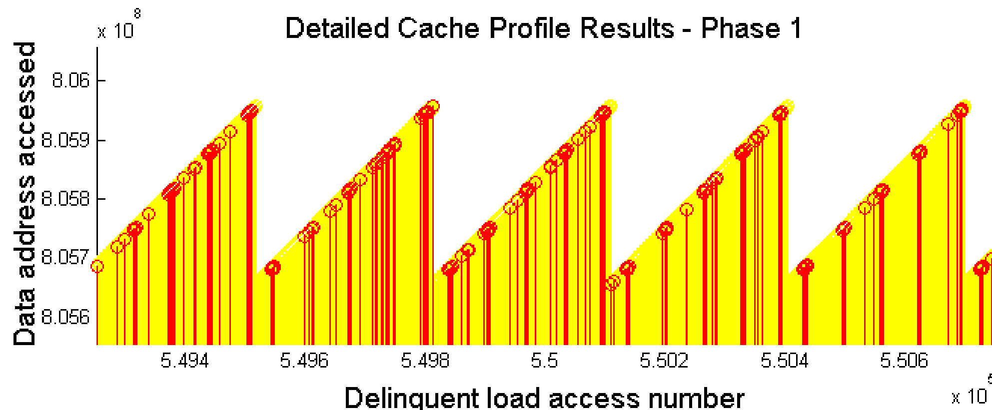
Detailed Benchmark Analysis

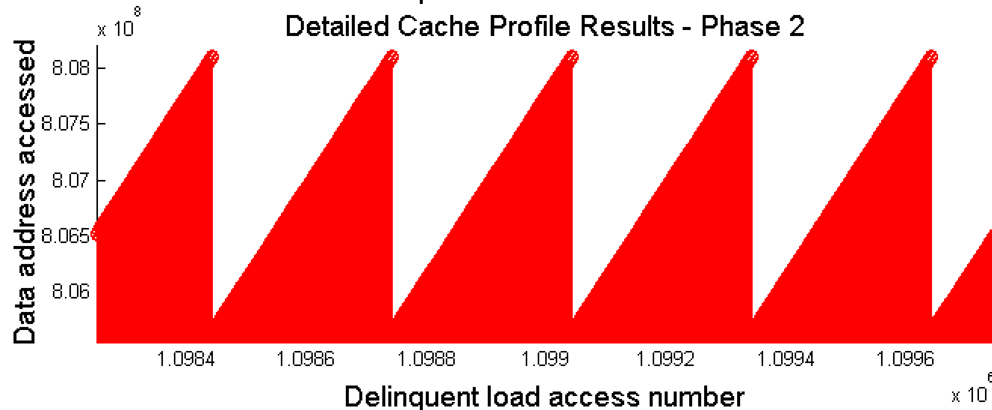
- The delinquent load traced to an iterative pointer access

```
181:     for( ; arc < stop_arcs; arc += nr_group )
  [pbeampp.c: 181] 0x2000f550:  414e03bb        cmpult  s1, s5, t12
  [pbeampp.c: 181] 0x2000f554:  e760002f        beq     t12, 0x2000f614
182:     {
183:         if( arc->ident > BASIC )
  [pbeampp.c: 183] 0x2000f558:  a42a0038        ldq     t0, 56(s1)
  [pbeampp.c: 183] 0x2000f55c:  2ffe0000        ldq_u   zero, 0(sp)
  [pbeampp.c: 183] 0x2000f560:  ec200027        ble     t0, 0x2000f600
184:         {
```

- Let’s examine this delinquent load in more detail
  - Use perfect branch prediction
  - This eliminates many speculative loads that are difficult to analyze

Detailed Benchmark Analysis





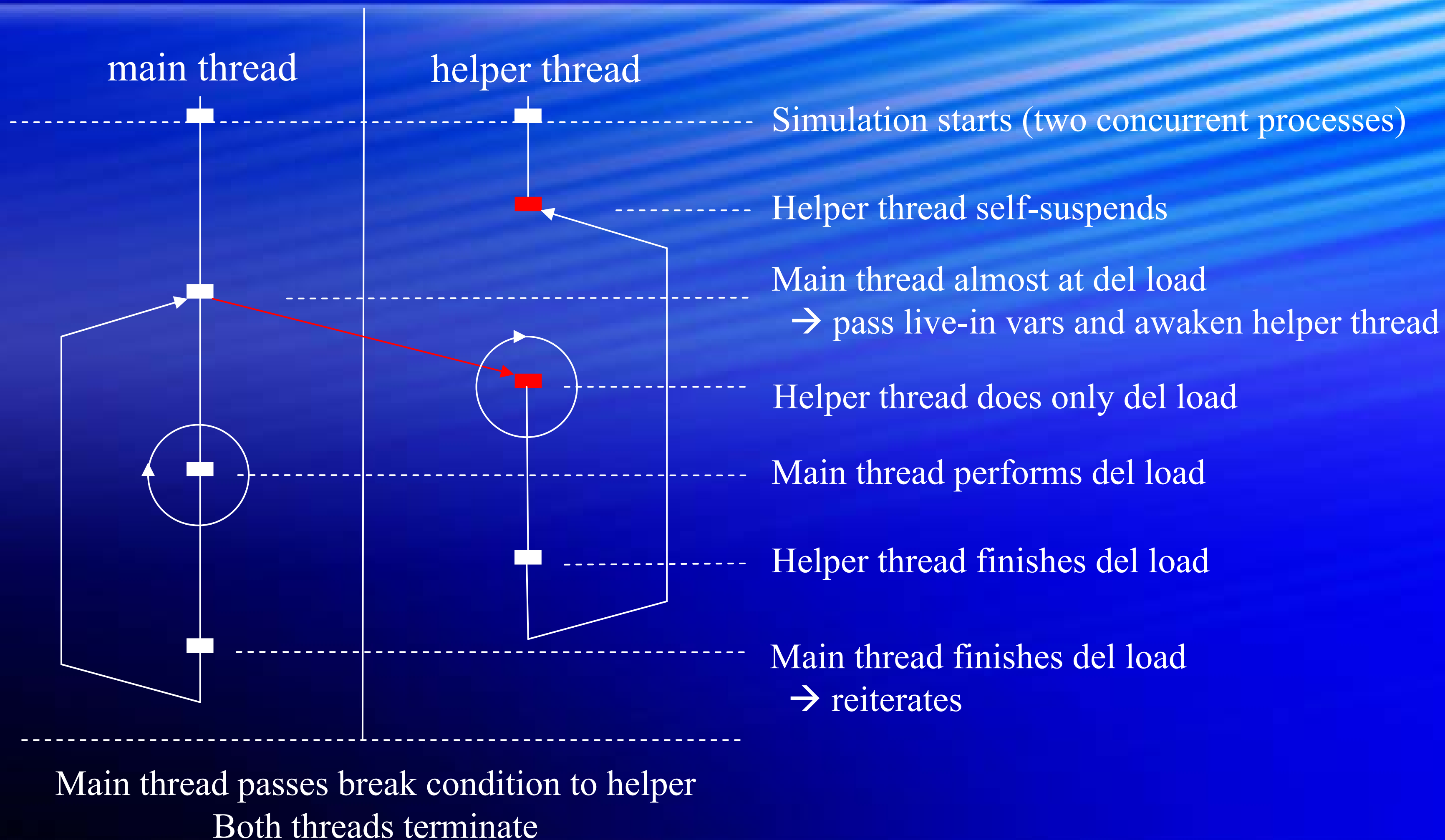
Legend

Green	L1 hit
Yellow	L2 hit
Red	L2 miss

Helper Process Motivation

- In the beginning, we figured out how to perform SMT by launching multiple processes at the beginning of simulation
- As a first step, let’s launch multiple processes, and allow one context to help the main context
- Allow inter-process communication via suspend/resume system calls and custom message send/receive instructions

Helper Process Flow Diagram



Helper Process Implementation

- Simulator Modifications:
  - Minor modifications in main to initiate the simulator
  - A major modification to the MMU
    - Cache tagging is based upon virtual address, not physical
    - Although both processes map an access to the same virtual address, cache lines are tagged with context ID. Hence, a major hack was necessary.
    - Major hack: If we are accessing the delinquent load from the helper thread, tag this load in cache as if it was loaded by the main context

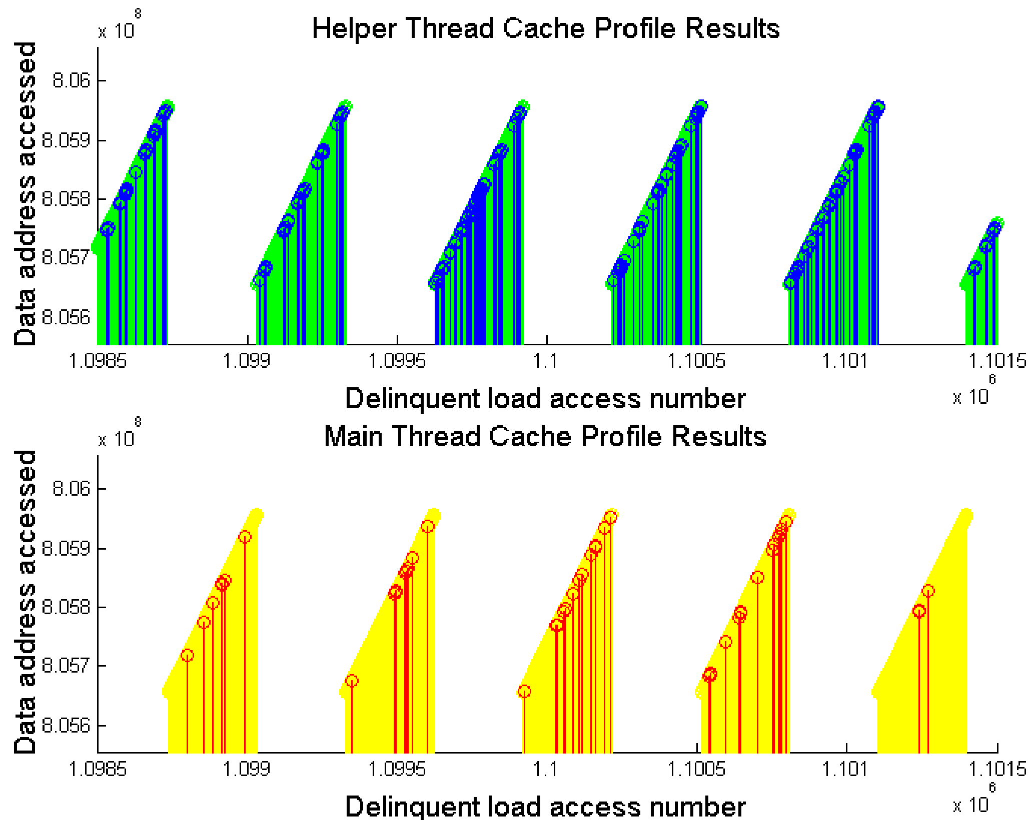
Helper Process Implementation

- Instruction Implementation:
  - Suspend/resume system calls were available, but required modification
    - Contexts were activated incorrectly
    - The resume command neglected to add 4 to the PC
  - Implemented new message-passing system calls for sending and receiving live-in variables between threads
    - GetPrivateDataSystemCall
    - SetPrivateDataSystemCall

Helper Process Implementation

- Other Modifications:
  - For the second phase of mcf, the helper thread was swapping values out of cache before they were used by the main thread
  - Solution 1: Reduce the number of iterations being helped such that thrashing no longer occurs
  - Solution 2: Implement dynamic synchronization between the two threads
  - At this point, solution 1 was implemented
  - Later, we implement solution 2

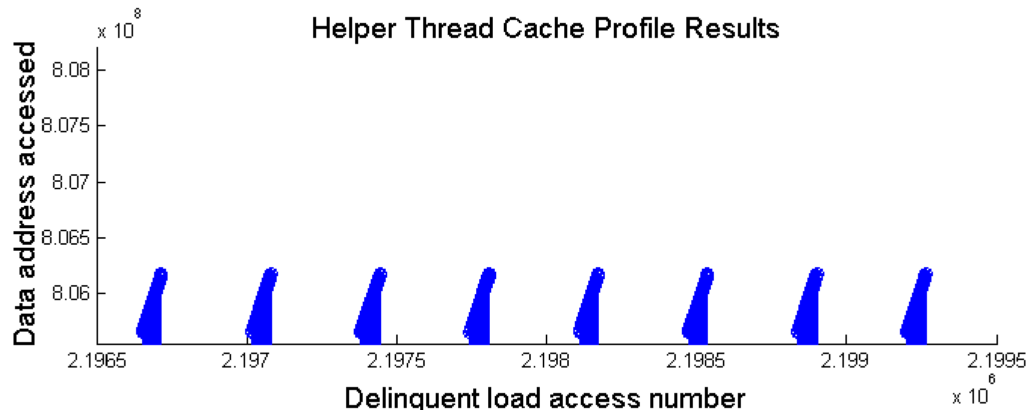
Helper Process Results (Phase 1)

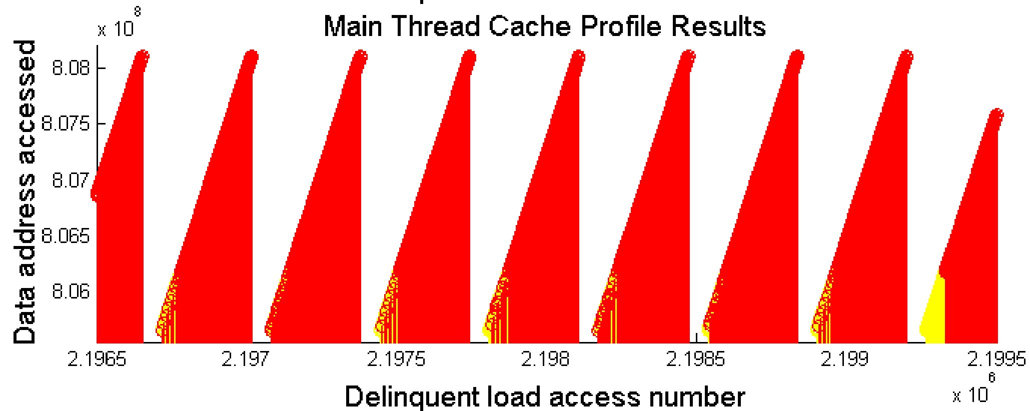


Legend

Green	L2 hit (H)
Blue	L2 miss (H)
Yellow	L2 hit (M)
Red	L2 miss (M)

Helper Process Results (Phase 2)



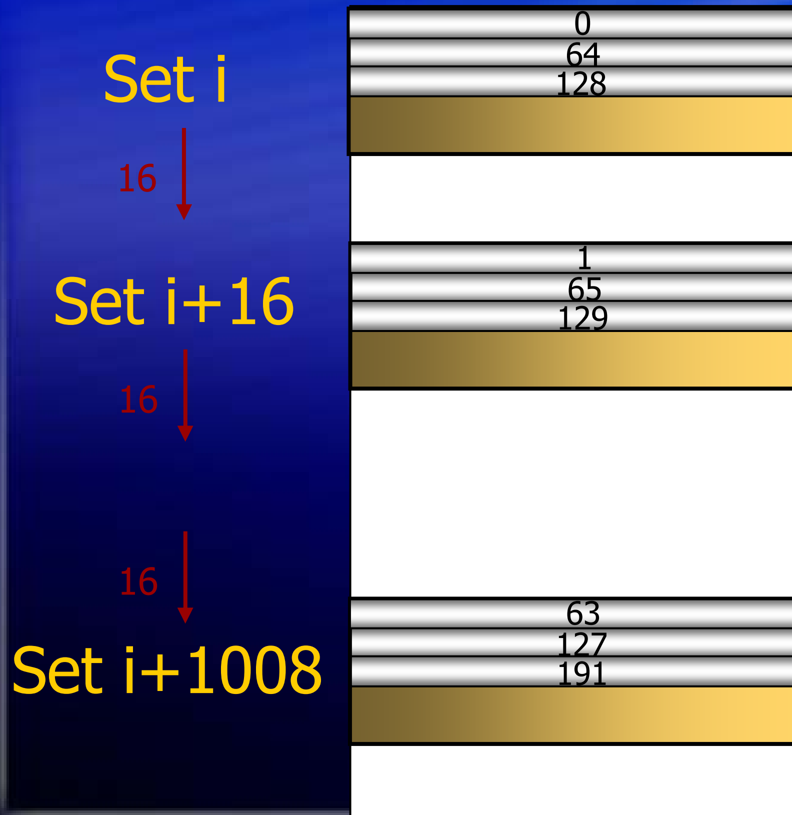


Legend

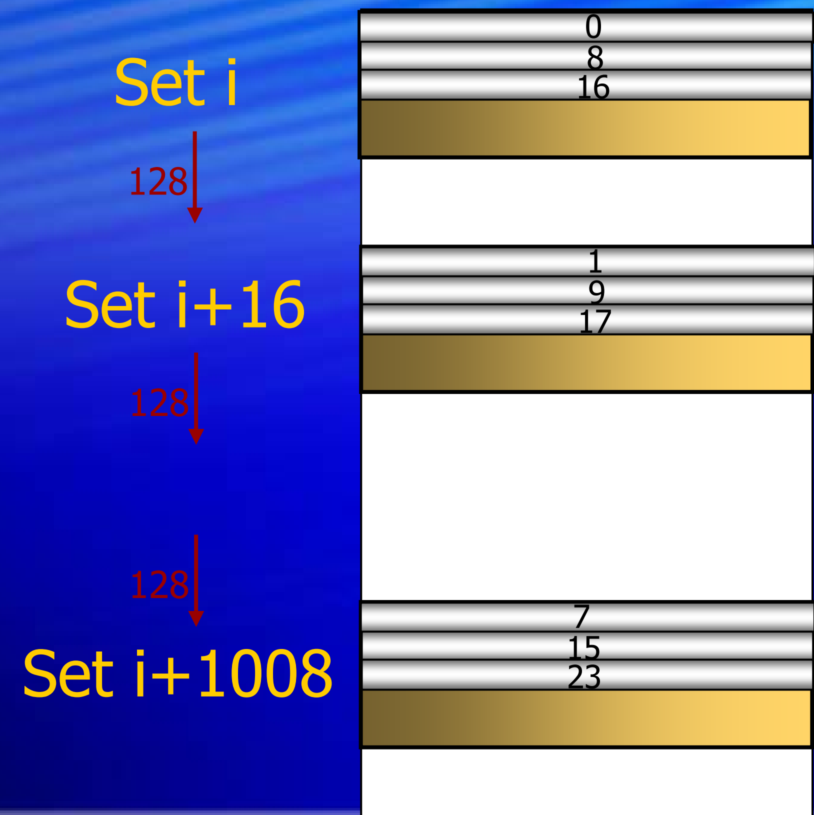
Green	L2 hit (H)
Blue	L2 miss (H)
Yellow	L2 hit (M)
Red	L2 miss (M)

Data Access Pattern

- Phase1: Advances by 1KB
  - Every 16 sets in L2
  - To previously accessed sets every 64 iterations
  - 8*64 best possible



- Phase2: Advances by 8KB
  - Every 128 sets in L2
  - To previously accesses set every 8 iterations
  - 8*8 best possible

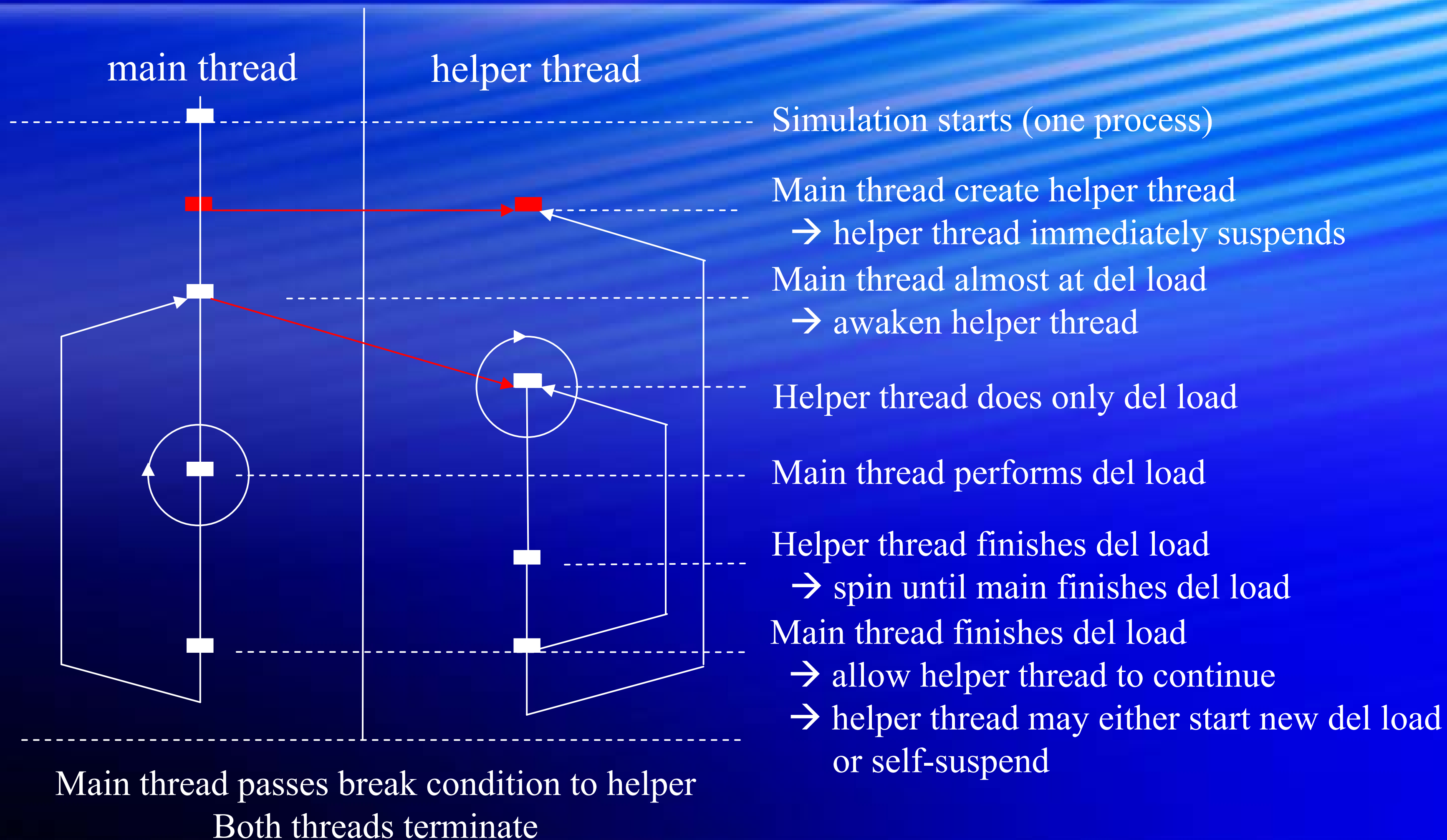


Helper Thread Motivation

- The previous solution was unrealistic
  - A cleaner solution would be to create a helper thread in main
  - Address space would be shared between the two contexts:
    - No need to hack the simulator's MMU
    - No need for inter-process message passing of live-in variables
    - Helper can monitor main thread's progress

      → dynamic synchronization

Helper Thread Flow Diagram



Helper Thread Implementation

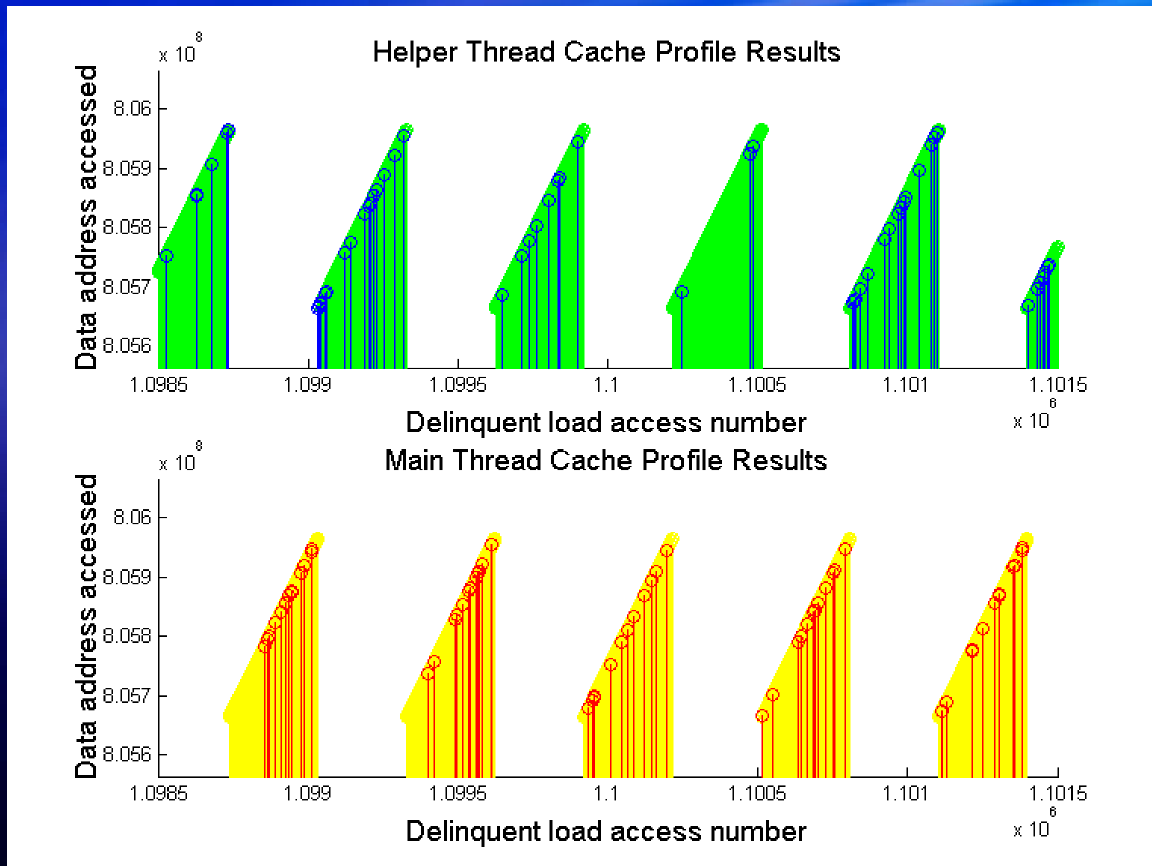
- Thread creation
  - Didn't work! Not activating the context properly.
- Simulator Modifications
  - Needed hacks to share cache, TLB, MMU between contexts
  - Another approach:
    - Using a dummy process to activate the second context

Thread creation

```
void _startThread()
{
    asm (".set    noreorder");
    asm("mov      0x1000,$sp") ;     //     # li      $sp,0x1000
    asm("perr     $sp,$sp,$18") ;    //     # gmsg    $sp,$sp
    asm("ldq      $16,-56($sp)") ;   //     # threadID
    asm("ldq      $17,-48($sp)") ;   //     # argv
    asm("ldq      $27,-40($sp)") ;   //     # function to execute
    asm("ldq      $29,-32($sp)") ;   //     # gp register
    asm("subq     $sp,96,$sp") ;     //
    asm("jsr      $26,($27)") ;      //     keeps return address in r26 and
    asm("ldq      $0,40($sp)") ;     //             jumps to r27 which is func
    asm("addq     $0,0x1000,$0") ;   //       # termination port
    asm("perr     $31,$31,$13") ;    //     # lcu     $0,$0 # endless wait
    asm (".set    reorder");
}

void createThread(__int64 threadId, __int64 context, __int64 stack,
          __int64 func, __int64 params)
{
   #define port 0x1000
  ((unsigned long long *)stack)[-7] = (unsigned long long)threadId;
  ((unsigned long long *)stack)[-6] = (unsigned long long)params;
  ((unsigned long long *)stack)[-5] = (unsigned long long)func;
  asm (".set    noreorder");
  asm ("stq $29, -32(%0)", stack);  // $29 is gp
  asm (".set    reorder");
  SSMT_LPCR(context, ((__int64)_startThread)  + 12);
  SSMT_PMSG(port, stack);
}
```

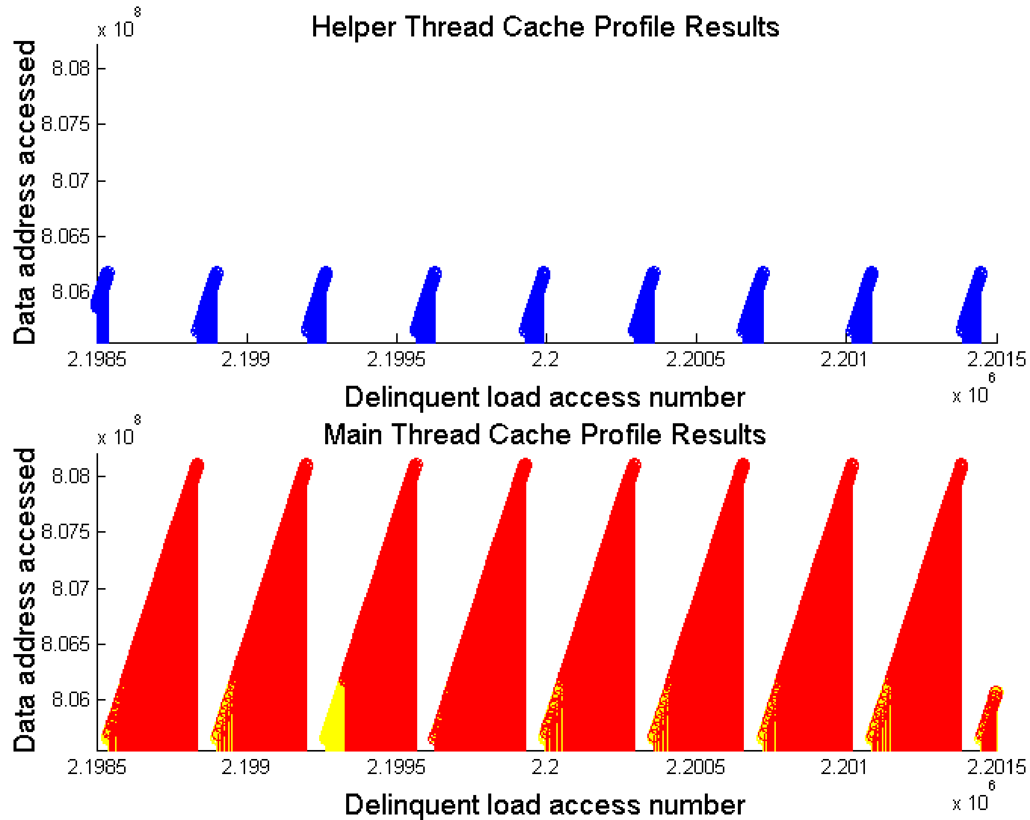
Helper Thread Results (Phase 1)



Legend

Green	L2 hit (H)
Blue	L2 miss (H)
Yellow	L2 hit (M)
Red	L2 miss (M)

Helper Thread Results (Phase 2)

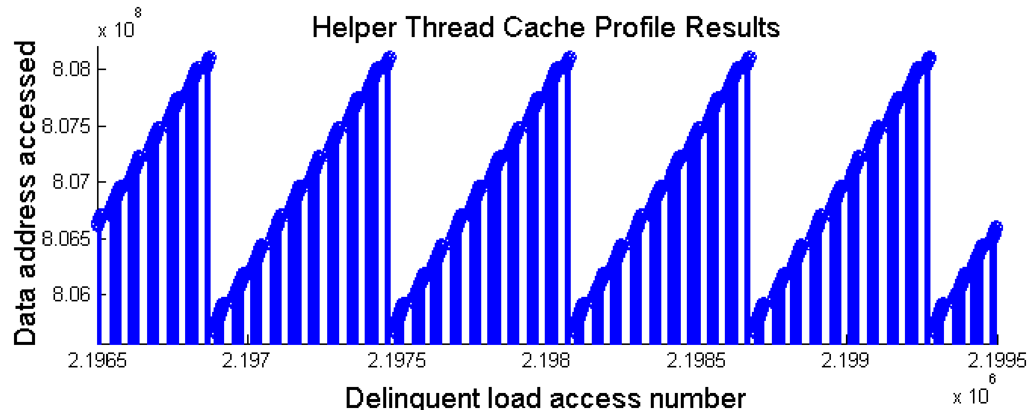


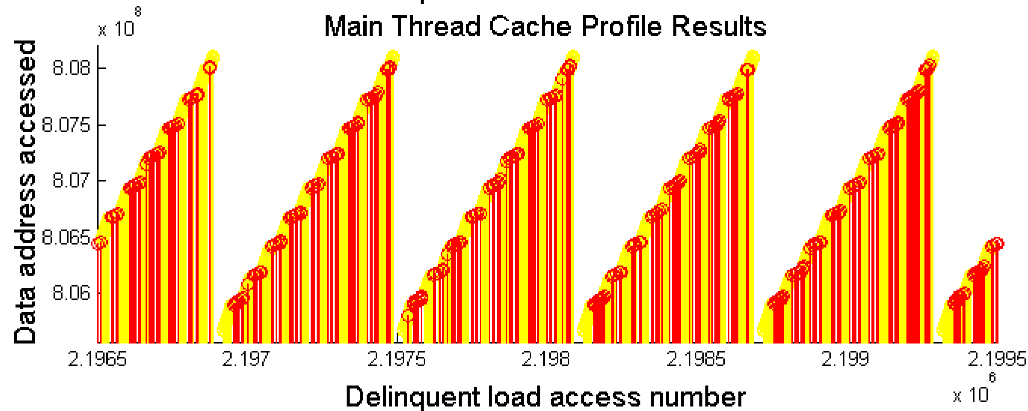
Static Synchronization

Legend

Green	L2 hit (H)
Blue	L2 miss (H)
Yellow	L2 hit (M)
Red	L2 miss (M)

Helper Thread Results (Phase 2)





Dynamic Synchronization

Legend

Green	L2 hit (H)
Blue	L2 miss (H)
Yellow	L2 hit (M)
Red	L2 miss (M)

Summary of Results

- Total Cycle count
- CL L1 hits
- CL L2 hits
- CL L2 misses
- For:
  - Original single-threaded program
  - Rev 1 (non-threaded, static synchronization with suspend/resume)
  - Rev 2 (threaded, single suspend/resume, dynamic synchronization based upon global variables)
  - Another (threaded, dynamic synchronization with suspend/resume

Conclusions

- Delinquent load performance was vastly improved (maximum of XXX % improvement)
- However, helper thread overhead and cache pollution negated any benefits on the delinquent load (maximum of XXX % improvement)
- Static synchronization performance was limited by thrashing
- Dynamic synchronization performance was limited because there is no sleep system call in the simulator
  - Either the main thread or the helper thread was frequently checking a condition
- Progress, in general, was limited by the lack of suitable documentation (most was in French)

References

[1] D. Kim, S. S. Liao, P. H. Wang, J del Cuvillo, X. Tian, X, Zou, H. Wang, D. Yeung, M. Girkar, J. P. Shen. Physical Experimentation with Prefetching Helper Threads on Intels Hyper-Threaded Processors. In *Proceedings of the Second Annual IEEE/ACM International Symposium on Code Generation and Optimization with Special Emphasis on Feedback-Directed and Runtime Optimization (CGO2004)*. San Jose, CA. March 2004.

[2] D. Madon, E. Sanchez, and S. Monnier. A Study of a Simultaneous Multithreaded Processor Implementation. In Euro-Par, August 1999.

[3] D. Berger, T. M. Austin. The SimpleScalar Tool Set, Version 2.0, *University of Wisconsin-Madison Computer Sciences Department Technical Report #1342*, June, 1997.

[4] SPEC. SPEC CPU2000 Benchmarks. Standard Performance Evaluation Corporation, 2000. http://www.spec.org/osg/cpu2000

[5] J. D. Collins, H. Wang, D. M. Tullsen, C. Hughes, Y.-F. Lee, D. Lavery, and J. P. Shen. Speculative Precomputation: Long-range Prefetching of Delinquent Loads. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 14-25, Goteborg, Sweden, June 2001. ACM.

[6] Intel Corporation. VTune Performance Analyzer.http://developer.intel.com/software/products/VTune/index.html.