# Exploitation of Nested Thread-Level Speculative Parallelism on Multi-Core Systems

Arun Kejariwal[§] Milind Girkar[‡] Xinmin Tian[‡] Hideki Saito[‡]
Alexandru Nicolau[†] Alexander V. Veidenbaum[†] Utpal Banerjee[†] Constantine D. Polychronopoulos[¶]
[§]Yahoo! Inc. [‡]Intel Corporation [†]University of California at Irvine [¶]University of Illinois at Urbana-Champaign

## ABSTRACT

*Multi-cores such as the Intel Core 2 Duo, AMD Barcelona and IBM POWER6 are becoming ubiquitous. The number of cores and the resulting hardware parallelism is poised to increase rapidly in the foreseeable future. Nested thread-level speculative parallelization has been proposed as a means to exploit the hardware parallelism of such systems. In this paper, we present a methodology to gauge the efficacy of nested thread-level speculation with increasing level of nesting.*

**Categories and Subject Descriptors**: C.4 [**Computer Systems Organization**]: Performance of Systems – Measurement techniques
**General Terms**: Performance, Measurement
**Keywords**: Thread-level speculation, performance

## 1. INTRODUCTION

Over the recent years there has been a proliferation in the number of cores per chip. For example, Intel and AMD have already fielded 6-core processors – "Dunnington" for Intel and "Istanbul" for AMD. The next few months will see the introduction of a raft of new x86 server chips that offer between 6 and 12 cores [1, 2]. Efficient use of such systems is critically dependent on the availability of concurrent software [3]. Thread-level speculation (TLS) has been proposed as one of the ways to parallelize difficult-to-analyze (potentially parallel) program regions. Although there has been a study on the efficacy of TLS at the innermost loop-level [4, 5] and call-graph level [6] evaluation of the performance potential of TLS, a limit study of the efficacy of nested TLS[1] has not been done.

In this paper we propose a methodology to assess the efficacy of nested TLS. We present a probabilistic model to determine the sensitivity of nested TLS with respect to misspeculation at the different levels of nested execution. Specifically, the parent-child relationship, a key aspect of nested speculative execution, is modeled using conditional probability. Our analysis shows that the efficacy of nested TLS decreases with increasing levels of nesting.

The rest of the paper is organized as follows: Section 2 overviews the different models for exploiting sTLP. The methodology of evaluation the efficacy of nested TLS is presented in Section 3.

## 2. BACKGROUND

In this section, we present an overview of the speculative execution model and the basics of conditional probability.

[1]The need for exploiting nested speculative thread-level parallelism (sTLP) [7] has its roots in the industrial trend of putting many (>2) cores on a single chip.


*CF'10,* May 17–19, 2010, Bertinoro, Italy.
ACM 978-1-4503-0044-5/10/05.

### 2.1 Speculative Execution Model

Existing TLS mechanisms can be broadly classified into two categories based on the speculative thread spawning policies: (a) in-order and (b) out-of-order (OOO) In the former, each thread can spawn at most one thread. On the other hand, in OOO spawn, a thread can spawn thread(s) in an unrestricted fashion. For example, let us consider the following loop (taken from `401.bzip2`, file name `compress.c`, line number 575):

```
L1: for (t = 0; t < nGroups; t++) {
S1:   Int32 curr = s->len[t][0];
S2:   bsW ( s, 5, curr );
L2:   for (i = 0; i < alphaSize; i++) {
L3:     while (curr < s->len[t][i]) { bsW(s,2,2); curr++; /* 10 */ };
L4:     while (curr > s->len[t][i]) { bsW(s,2,3); curr--; /* 11 */ };
S3:     bsW ( s, 1, 0 );
      }
    }
```

Figure 1(a) illustrates the in-order spawn model wherein the non-speculative thread $T_0$ spawns speculative thread $T_1$ which in turn spawns speculative thread $T_2$ and so on. In contrast, in OOO spawn, see Figure 1(b), non-speculative thread $T_0$ spawns two speculative $T_1$ and $T_2$.[2] Thread $T_2$ then spawns speculative threads $T_3, T_4$ and $T_5$. Thread $T_0$ executes the statement `S1` and the function `bsW` (`S2` in the snippet) non-speculatively, whereas the thread $T_5$, for example, speculatively executes the function `bsW` inside the inner `for` loop. Clearly, OOO spawn exposes higher degree of sTLP. Therefore, we assume an OOO spawn policy for the analysis presented in this paper.

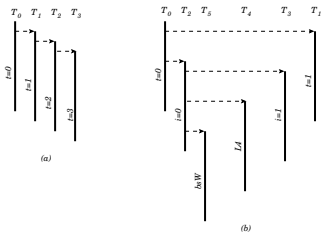

**Figure 1: (a) In-order spawn (b) Out-of-order spawn**

### 2.2 Conditional Probability

Let `A` and `B` be two events. The conditional probability of event `A` to occur, given that event `B` has occurred, is given as follows:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (1)$$

where, $P(B)$ denotes the probability of occurrence of event `B` and $P(A \cap B)$ denotes the occurrence of both the events `A` and `B`.

[2]Due to the space limitations, speculative spawning of other threads corresponding to the iterations $t = 2, \ldots,$ nGroups is not shown.

## 3. NESTED TLS

One way to leverage the large number of cores (on a chip) is to exploit nested sTLP. Several techniques have been proposed for this. However, none of the existing techniques has evaluated the sensitivity of the profitability of nested TLS w.r.t. speculation depth. This makes it difficult to gauge the performance gain achievable via nested TLS. To this end, we propose an analytical model based on conditional probability which captures the parent-child relationship – a key aspect of nested TLS. We illustrate the model with the help of an example call graph, shown in Figure 2. The function `Strparse` is part of the `456.hmmer` benchmark.

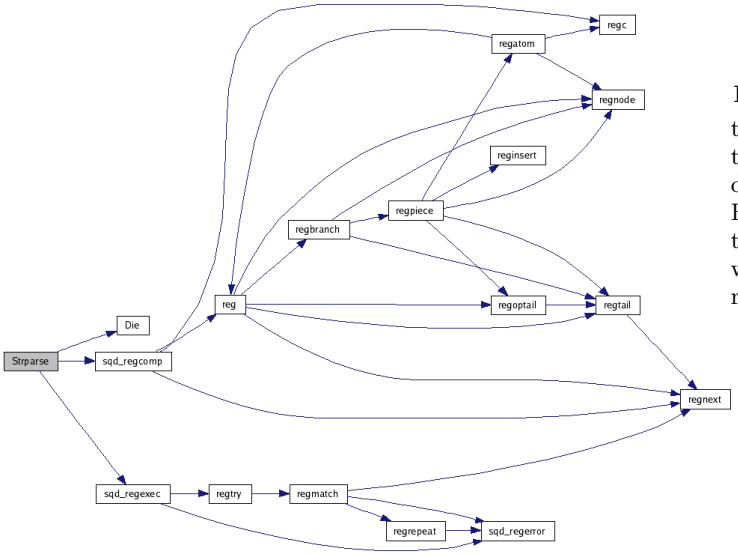


**Figure 2: An example call graph with a depth 7**

From the figure we see that the call graph of `Strparse` has a depth of 7. Let us consider the path along `Strparse, sqd_regcomp, reg, regbranch, regpiece, regatom` and `regc`. Assume all the functions along the path are executed speculatively. Then, the successful execution of `regc`, denoted by event `A`, is contingent on the successful execution of `regatom`, denoted by event `B`. The variation of $P(A|B)$, i.e., probability of successful execution of `regc` given that `regatom` was successfully executed, is shown in Figure 3. Note that $P(A \cap B) \leq P(B)$.

From the figure (along the $P(B)$ axis) we note that $P(A|B)$ decreases linearly with decrease in $P(A \cap B)$. $P(A \cap B)$ corresponds to the case wherein no conflicts occur between the speculative threads executing `regatom` and `regc`. Even if $P(A \cap B) = 1$, this does not, by itself, guarantee an overall correct speculation as the success of `A` and `B` is dependent on the correct speculation of the function `regpiece`, denoted by event `C`. To address this, we extend the model to capture higher levels in of the call graph. The probability of success of `B` given that `C` was successfully executed is given by:

$$P(B|C) = \frac{P(B \cap C)}{P(C)}$$

In a similar fashion, the model can be extended to all 7 levels of speculation. The overall probability of correct speculation decreases exponentially with increase in the speculation nesting level.

We analyze the hottest loop in `435.gromacs` [8], taken from `innerf.f:3932` (see Figure 4), as a case study. The loop is doubly nested and covers 57% of the total execution time. The loop nest contains many array references through subscript arrays (e.g., `faction(jjnr(k)-1)`), thus dependences on this loop nest cannot be detected statically. Besides the reduction updates to elements of `faction` between line 4140 and 4148, there are additional reads and writes to `faction` in the inner k-loop that are not in the reduction form.

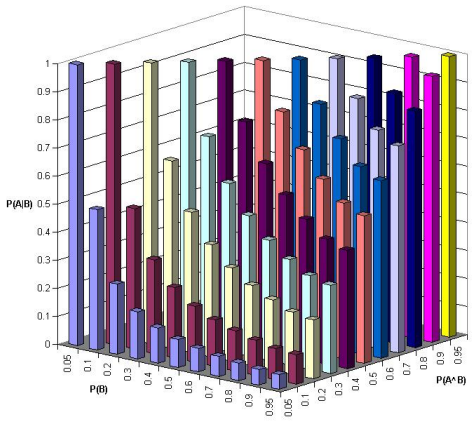


**Figure 3: Variation of $P(A|B)$ (refer to Equation 1)**

```
3932   do n=1,nri
         ...
         ii3 = 3*iinr(n)-1
         ...
3961     do k=nj0,nj1
           jnr = jjnr(k)+1
           j3  = 3*jnr-2
           ...
           fjx1 = faction(j3)-tx11
           ...
           faction(j3)   = fjx1-tx31
           ...
4139     end do
4140     faction(ii3)     = faction(ii3) + fix1 /* dep freq 51% */
4141     faction(ii3+1)   = faction(ii3+1) + fiy1 /* dep freq 51% */
         ...
4148     faction(ii3+8)   = faction(ii3+8) + fiz3 /* dep freq 51% */
         ...
4155   end do
```

**Figure 4: Hot loop at `435.gromacs:innerf.c:3932`**

In [9], Wu et al. showed that, with the training input data set, the inner loop is profiled to be parallel and is thus amenable for TLS. The frequency of run-time materialization of dependences associated with `faction` is shown in Figure 4 (taken from [9]). True dependences associated with the references to `faction` in the inner and the outer loop results in degrading the efficacy of nested TLS by 50%. Similar observation, i.e., decreasing efficacy of nested TLS with 2 or more nesting levels, was made for other applications in the SPEC CPU2006 suite such as `429.mcf`.

## 4. REFERENCES


[1] AMD Confirms 12-Core Opteron Production, Februrary, 2010. http://www.hpcwire.com/blogs/AMD-Confirms-12-Core-Opteron-Production-85118242.html#38593146.

[2] Multicore Watershed, March, 2010. http://www.hpcwire.com/blogs/Multicore-Watershed-86426622.htm.

[3] H. Sutter and J. Larus. Software and the concurrency revolution. *ACM Queue*, 3(7), 2005.

[4] A. Kejariwal, X. Tian, W. Li, M. Girkar, S. Kozhukhov, H. Saito, U. Banerjee, A. Nicolau, A. V. Veidenbaum, and C. D. Polychronopoulos. On the performance potential of different types of speculative thread-level parallelism. In *Proceedings of the 20th ACM ICS*, pages 24–35, 2006.

[5] A. Kejariwal, X. Tian, M. Girkar, W. Li, S. Kozhukhov, H. Saito, U. Banerjee, A. Nicolau, A. V. Veidenbaum, and C. D. Polychronopoulos. Tight analysis of the performance potential of thread speculation using SPEC CPU2006. In *Proceedings of the 12th ACM SIGPLAN Symposium on PPoPP*, 2007.

[6] A. Kejariwal, M. Girkar, X. Tian, H. Saito, A. Nicolau, A. V. Veidenbaum, and U. Banerjee. On the efficacy of call graph-level thread-level speculation. In *Proceedings of the First Joint WOSP/SIPEW ICPE*, pages 247–248, 2010.

[7] J. Renau, J. Tuck, W. Liu, L. Ceze, K. Strauss, and J. Torrellas. Tasking with out-of-order spawn in TLS chip multiprocessors: Microarchitecture and compilation. In *Proceedings of the 19th ACM ICS*, pages 179–188, 2005.

[8] SPEC CFP2006. http://www.spec.org/cpu2006/CFP2006.

[9] P. Wu, A. Kejariwal, and C. Caşcaval. Compiler-driven dependence profiling to guide program parallelization. In *Proceedings of the 21st International Workshop on LCPC*, 2008.