Speculative Hedge: Regulating Compile - Time Speculation Against Profile Variation

INTRODUCTION

Scheduling Heuristics:

• *Dependence height*

❁ widely used in trace and super-block scheduling

❁ (-) over-speculation along predicted important path may lead to unnecessary delay for seemingly unimportant path

• *Dependence height & speculative yield*

❁ (+) pay attention to the needs of all paths

❁ (-) but does not address the mismatches between prediction and run-time behavior

• *All resource (including dependence) height & speculative yield*

(Speculative Hedge) --- focus of this paper

❁ (+) overcome the drawbacks of the previous two

❁ (-) does not solve what paths should be scheduled together

BACKGROUND

- *Late times*

❁ the latest time that an operation can be scheduled without delaying an exit

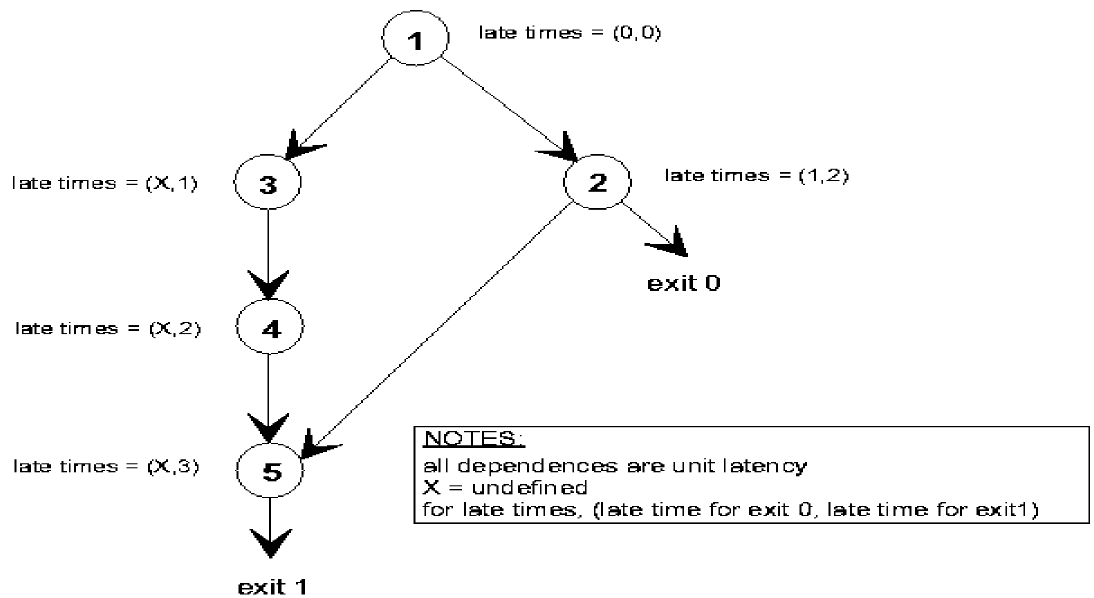


Figure 1. Dependence graph used for scheduling

❁ extends from just accounting for dependence to all resource heights

BACKGROUND (cont.)

- *Dependence height & speculative yield*

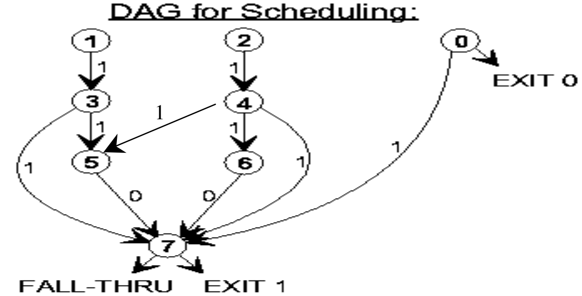
  - $$Priority_x = \sum_{ValLT_x} (Prob_i * (MaxLT + 1 - LT_x))$$
    $$ValLT_x = \text{for\_all\_valid\_late\_times}_i\text{\_of\_op}_x$$
    $$Prob_i = \text{probability\_of\_taking\_exit}_i$$
    $$MaxLT = \text{max\_late\_time\_in\_dep\_graph}$$
    $$LT_x = \text{op}_x\text{\_late\_time\_for\_exit}_i$$

  - Using only dependence height can cause delay unnecessarily

Operation Sequence:

```
0:  branch r10==0, target0
1:  r1 <- ld(r2 + 0)
2:  r3 <- ld(r2 + 4)
3:  r4 <- r1 + r5
4:  r6 <- r3 + r7
5:  r8 <- r6 * r4
6:  r9 <- r6 * r10
7:  branch r4==r6, target1
```

DAG for Scheduling:



Notes:

1) 2-issue processor which can issue 1 branch per cycle
2) 0-cycle dependences needed to ensure that operations that generate a live-out value are kept above their branch
3) dependence height = 3
4) height based on issue width = ceil(8/2) = 4

Figure 2. Example of dependence height not being a problem.

BACKGROUND (cont.)

More Scheduling Heuristics:

- *Critical path*

  ❋ use dependence height associated with the last exit in a superblock

  ❋ *Dependence height & speculative yield* priority function creates identical schedules to *critical path* if profile information shows the last exit is always taken

  ❋ (-) same problem as *Dependence height & speculative yield* exists

- *Successive retirement*

  ❋ attempts to retire each exit in-order, as early as possible

  ❋ only speculates when there are no non-speculative instructions available, so minimizes speculation

  ❋ (+) works well as speculative hedge for narrow-issue processors, where speculation is not very important

  ❋ (-) not enough speculation for wide-issue processors

SPECULATIVE HEDGE

High-Level Overview:

• *Dynamic-priority scheme*

❋ different resources constraints at various times during scheduling

❋ scheduling decision affected by the type of resources constraints

• *Priority values*

❋ Helped weight - the sum of the taken probabilities for all exits having a *critical need* met by an operation --- the only one which depends on profile information among all the four priority values

❋ Helped count - the total number of exits having a *critical need* met by an operation

❋ Minimum late time difference - the minimum difference between the current cycle time being scheduled and any of an operation's late times

❋ Original program order - a unique number which is based on the original program location of an operation

SPECULATIVE HEDGE (cont.)

- *Observations*

  ❈ the next priority value is used when the previous one(s) have tied

  ❈ insensitive to profile information

  ❈ speculative hedge does not keep exits from being delayed because of speculation. Nevertheless, it minimizes the number of exits and penalty for exits that are delayed unnecessarily

- *Procedure*

  ❈ first, the retirement goals (the earliest time based on certain resources constraint) and critical needs for every exit that has not been retired are determined

  ❈ Then, all the late times for unscheduled operations are computed using the exit retirement goals as the defined late times for their corresponding exit operation

  ❈ Finally, the priority values for all operations available for scheduling are computed, the operation with the highest priority gets scheduled

SPECULATIVE HEDGE

Exit retirement goals and Critical needs:

- First, compute *Cycle estimate* for each exit based exclusively on
  - *dependence height*
  - *branch unit*
  - *issue width:* $ce_{iw} = \lceil num\_ops\_to\_sched\_before\_exit \,/\, issue\_width \rceil$

    it does not detect all issue-width problems though, as this shows

Figure 3. Example of an issue-width problem located at the beginning of a dependence chain.

  - *and all other applicable resources constraints*

- Then, take the maximum of all the cycle estimates for each need yields the retirement goal for an exit. Any need whose cycle estimate equals the exit retirement goal is considered critical

SPECULATIVE HEDGE

Priority Calculation:

```
for (each unscheduled, available oper_x) {
   helped_weight_x = helped_count_x = 0
   for (each exit_i) helped_found_i = FALSE
   for (each exit_i where oper_x has a valid late time_i) {
      if (dependence_critical_for_i &&
         (oper_x's late time_i == current_cycle)) {
            helped_found_i = TRUE
            account_for_future_branches
      }
      if (issue_width_critical_for_i &&
         oper_x's late time is located before issue width
         problem ends) {
            helped_found_i = TRUE
            account_for_future_branches
      }
      /* the following is an example of what needs to be
            done for restricted resources */

      if (integer_alu_issue_width_critical_for_i &&
         oper_x is an integer alu operation &&
         oper_x's late time is located before integer alu
         issue width problem ends) {
            helped_found_i = TRUE
            account_for_future_branches
      }
   }
   for (each exit_i)
      if (helped_found_i) {
         helped_weight_x += profile_weight_i
         helped_count_x++
      }
   determine_min_late_time_diff_for_oper_x
}
```

Figure 4. Algorithm for dynamic priority calculation.

SPECULATIVE HEDGE

Priority Calculation (cont.):

• basically, it is performed for every operation that can be scheduled by evaluating each exit, and determining whether the operation meets a critical need for the exit

• a check is also made to determine if the meeting of a critical need for the current $exit_i$ will help future exits meet a critical need, especially in the multiple branch case, e.g.,

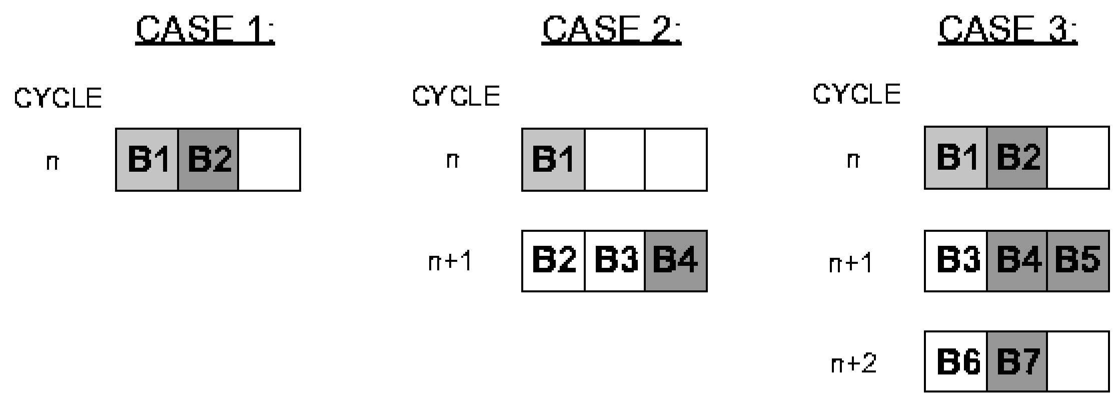


Figure 5. Cases for account_for_future_branches.

SPECULATIVE HEDGE

Potential Problem:

basically NP-complete problem, obtaining an optimal solution for all cases is not feasible

• speculative hedge only looks at the situation for the current cycle, it may not always make the best decisions for the entire scheduling region as a whole

• speculative hedge only looks at the cost involved for the current cycle, and

does not look at the costs that are involved in the future

• it does not consciously schedule for an exit's retirement until it is absolutely necessary to do so

EXPERIMENT RESULT

Table 2. Results when input data matches input used for REAL profiling.

Machine Model	Benchmark	SH - REAL # of cycles	SH - ALL1 % diff	SH - LAST1 % diff	DHASY - REAL % diff	DHASY - ALL1 % diff	Crit. Path[a] % diff	Succ. Ret. % diff
4_MIX	espresso	152405400	-0.5	-0.9	-0.2	-0.6	-2.1	-0.7
	li	14486519	0.0	0.0	0.0	-0.7	-0.3	-0.4
	eqntott	423624658	0.0	0.0	0.0	0.0	0.0	0.0
	compress	25767022	-0.1	-0.1	-0.6	-3.1	-8.2	-2.6
	sc	38204165	0.0	0.0	-0.5	-0.9	-0.3	-3.7
	cc1	49349343	-0.2	-0.1	-0.3	-0.7	-0.2	-1.1
4_1BR	espresso	146674421	-0.1	0.0	-1.4	-1.2	-1.0	-1.6
	li	13702023	0.0	0.0	-0.8	-1.3	-0.2	-1.1
	eqntott	401064104	0.0	0.0	0.0	0.0	0.0	0.0
	compress	24174638	-0.5	-0.5	-0.4	-3.4	-6.3	-2.9
	sc	36938586	0.0	0.0	-0.1	-0.3	0.0	-2.5
	cc1	47497736	0.0	0.0	-0.1	-0.2	-0.1	-0.7
4_2BR[b]	espresso	129275735	-0.1	0.0	-1.0	-1.9	-7.2	-2.3
	li	12389729	-0.7	-0.7	0.0	-2.2	-1.3	-2.0
	eqntott	313622471	-0.4	-0.4	-1.0	-4.0	-6.4	-0.9
	compress	22011735	-0.3	-0.3	-0.6	-7.1	-11.4	-2.7
	sc	30904853	-1.2	-1.1	-1.4	-2.0	-5.8	-4.6
	cc1	41450987	0.0	0.0	-0.4	-0.6	-3.7	-1.3
SINGLE	espresso	330192736	-0.2	-0.2	-1.6	-3.1	-11.2	-0.2
	li	27242181	-0.6	-0.6	-0.3	-1.9	-4.4	-1.9
	eqntott	826767804	0.0	0.0	0.0	-0.1	-0.1	-0.1
	compress	56980582	-2.0	-2.0	-0.7	-6.3	-15.6	+0.1
	sc	84079061	-0.5	-0.4	+0.2	-0.8	-1.0	-0.7
	cc1	99468982	-0.3	-0.2	-0.1	-1.4	-3.1	-0.4

[a] Critical path is the same as DHASY-LAST1.
[b] All branch-to-branch dependences for DHASY are assumed to be 0.

EXPERIMENT RESULT(cont.)

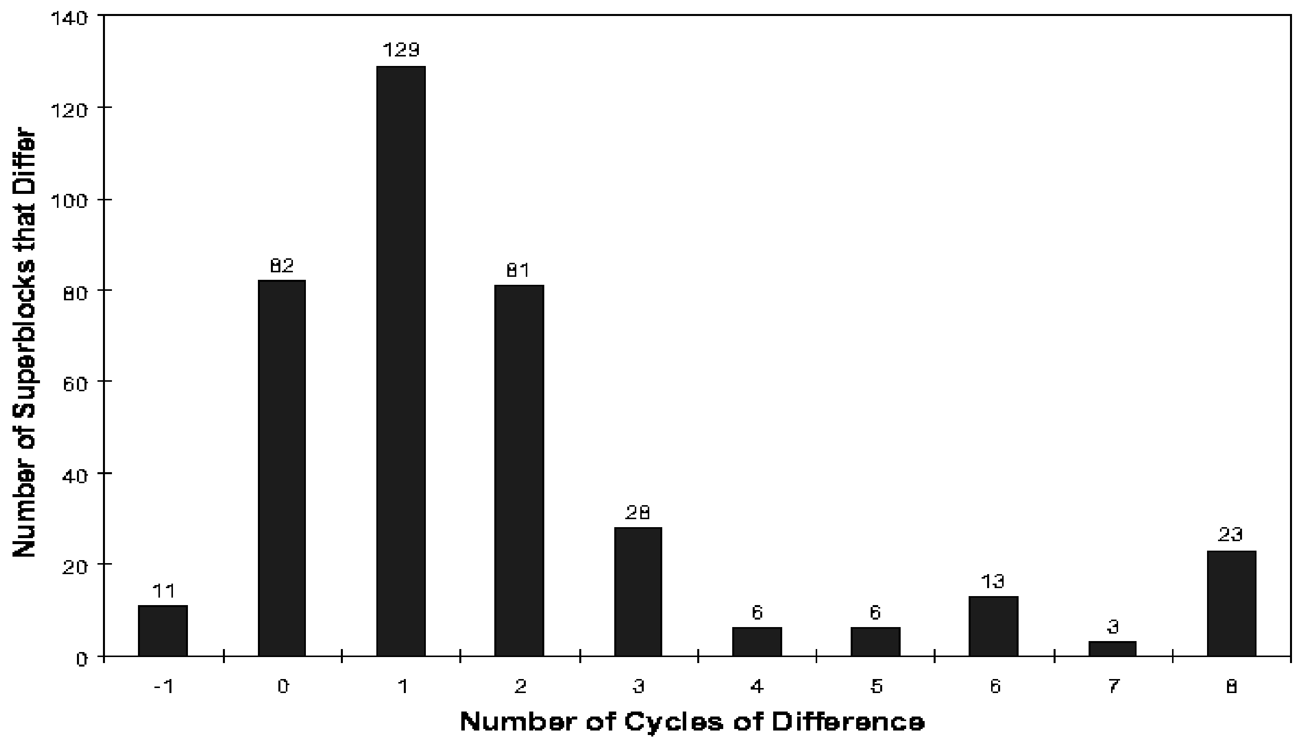


Figure 6. Cycle differences between SH-LAST1 and DHASY-LAST1 for the first exit.

EXPERIMENT RESULT(cont.)

For six program from the SPEC CINT92 benchmark suite for four different machine models with three different sets of compile-time predicted behaviors

- The largest variation for any combination of machine model and compile-time predicted behavior was 2% for SH; while DHASY, using the same set of compile-time predicted behavior, had variations of over 5% for at least one benchmark in every machine model, with three cases having variations of over 10%
- The analysis of how the first exits of superblocks can be delayed for different profile inputs showed that DHASY dealyed some superblock first exits by up to eight cycles more than speculative hedge

SUMMARY

- Speculative hedge attempts to minimize speculation so paths of execution do not get delayed unnecessarily
- This ensures that speculative hedge is much less sensitive to variations between compile-time predicted behavior and actual run-time behavior