Do Inputs Matter? Using Data-Dependence Profiling to Evaluate Thread Level Speculation in BG/Q

Arnamoy Bhattacharyya
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
Email: arnamoy@ualberta.ca

I. INTRODUCTION

Thread level speculation (TLS) is a hardware/software technique that guarantees correct execution of a loop even in the presence of a dependence. To reduce mis speculation overhead, data-dependence profiling is used to find out whether the may dependences materialize during runtime. Based on the probability of dependence, a cost model can be used to select candidate loops for speculative execution. But a single input profile is not sufficient to capture the dependence behaviour of a loop because Berube et al. showed that programs’ behaviour may change based on input [2]. Though previous work mentions that there is little variability in the dependence behaviour of loops based on inputs [1], there has not been an extensive study to support the claim.

II. RESEARCH CONTRIBUTION

- 57 benchmarks from four different benchmarks suites (SPEC2006, PolyBench/C, BioBenchmark, NAS) that were used in TLS research before, are studied to verify whether the previous claim on the invariability of loops’ dependence behaviour based on inputs is true. Results show that the the dependence behaviour does not change based on inputs in the speculation candidate loops in these benchmarks.

- Based on this finding, single input data-dependence profiles are used to evaluate the performance of TLS implementation in the IBM’s BlueGene/Q (BG/Q) supercomputer, where the inputs to the profiling run and training run are different. This work is the first performance evaluation of the TLS implementation in BG/Q.

- A detailed study on the different factors that have impact on TLS performance is performed. These factors include: (1) number and coverage of speculative loops; (2) mis speculation overhead due to runtime dependences introduced by function calls inside the loop body; (3) L1 cache miss increase due to the long-running (LR) mode in BG/Q; and (4) dynamic instruction path length increase in loops that have small iteration counts but form significant code portion of a benchmark.

III. RESULTS

Figure 1 shows the performance of three parallel versions (auto-SIMDized, auto-SIMDized+auto-OpenMP) by bgxlc_r

Fig. 1. Speedup of different parallel versions of SPEC2006 and PolyBench/C Benchmarks.

and auto-SIMDized+auto-OpenMP+speculatively parallelized by an automatic speculative parallelization framework developed of the SPEC2006 and PolyBench/C benchmarks. The speculative loops in ibm have 98% coverage that accounts for the speedup while in bzfp2 (35%) and dynprog (26%), the poor coverage of speculative loops introduces overhead. h264ref has the highest number of loops speculatively parallelized (47) but most of them have function calls that introduce dependences, thus causing slowdown (only 12% of speculative threads successfully committed). Filtering speculative execution of loops with non-side-effect-free function calls tackles the mis speculation overhead. cholesky and dynprog experience L1 cache misses due to LR mode(12% and 10% respectively) while jacobi and seidel experience huge dynamic path length increase (112% and 123% respectively over sequential).

REFERENCES
