BBC: A Basic Block Compiler for Mitigating Branch Miss Prediction Penalty in Mobile MPSoCs

Kyu Jung, Ph.D.
Adaptmicrosys

Paul Jung
Adaptmicrosys

***Abstract*—This paper proposes a basic-block fetch paradigm that improves program control flow by fetching flow-control instructions first, followed by other instructions in a timely and accurate manner. By predicting multiple basic blocks early and fetching instructions within fewer cycles, the instruction-fetch paradigm achieves high-bandwidth accurate fetch through out-of-order and parallel fetching. It also proves more resilient to branch prediction and instruction cache miss latencies, resulting in significant bandwidth enhancements for both out-of-order and in-order CPUs.**

***Keywords*—Basic Block Compiler; Lookahead branch prediction; Instruction fetch bandwidth, High Performance Computing (HPC) Performance and Low-power Operations**

I. Introduction

This paper proposes an instruction fetch (i-fetch) paradigm to address the high-bandwidth instruction fetching challenge for modern out-of-order (OoO) CPUs. The paradigm reduces the i-fetch window and instruction cache (i-cache) block sizes, uses multiple branches for dynamic control flow determination, and fetches instructions from different basic blocks in parallel. The paper also introduces a basic block compilation (BBC) technique to transform control-flow programs into decoupled programs. The proposed i-fetch paradigm outperforms existing sequential and parallel frontend CPUs in i-fetch bandwidth and power consumption.

II. Basic Block Compilation (BBC) Technique

A program with control flow is made up of sequences of basic blocks, which consist of variable-size contiguous instruction blocks with flow-control instructions often located at the ends. The variable size of these basic blocks and the taken branches at their ends make it difficult to efficiently use i-cache space and can lead to wasted energy in i-fetch hardware. Accessing a fixed number of contiguous instructions (e.g., 16/8/4 instructions) is inherently limiting due to the variable size of the basic blocks. Moreover, predicting branch instructions of basic blocks must be done within one cycle to avoid additional inefficiencies, such as the power consumption of the branch prediction unit (BPU). One possible solution is to fetch the branch instruction located at the end of a basic block before or at the same time as the other contiguous instructions of the basic block.

III. Flow Control Decoupling for Branch Prediction

The program counter in a control-flow program tracks control flow at the instruction level. To improve performance, flow-control instructions can be decoupled from basic blocks, thereby converting fine-grained instruction-level control flow into coarse-grained basic-block-level control flow. This transformation enables decoupled flow-control instructions to be fetched at least one cycle ahead or at the same cycle as the contiguous instructions of the associated basic blocks. This approach eliminates multi-cycle prediction latency and reduces power consumption of the BPU. Previous studies have shown that equivalent or higher prediction accuracies can be achieved using predictors for the coarse granularity of control flow. Decoupling flow-control instructions during compilation simplifies parallel instruction fetching and enables the use of low-power instruction caches for out-of-order CPUs. This approach is simpler than the mechanism in previous research, which involves multiple instruction sequencers, fragment buffers, and renaming units.

IV. Decoupling Flow-control Instructions for Parallel Fetch

The BBC technique splits programs into two subprograms: a decoupled control-flow subprogram and a functional subprogram. Flow-control instructions in basic blocks are modified to redirect control flow to target locations and access contiguous instructions in the functional subprogram. Non-flow-control instructions can be added for parallel fetching by converting large basic blocks to fragments or using basic blocks without flow-control instructions. Instructions in the decoupled control-flow subprogram provide access points to the associated whole or fragmented basic block. Only necessary flow-control instructions are fetched in a sequential or parallel manner, unlike the conventional i-fetch paradigm.

V. Lookahead Out-of-Order Parallel I-Fetch

In contrast to a sequential fetch paradigm, a parallel fetch paradigm fetches blocks of contiguous instructions from multiple instruction streams out of order. However, the flow-control instructions are fetched in the order they are in the program. Despite the advantages of out-of-order fetch, parallel fetch still has limitations for wide-issue OoO CPUs due to the complexity and overhead of the i-fetch hardware, multiple interrupt/exception services, and power consumption of BPUs. In addition, large wrong-path instructions increase i-cache pollutions and misses. While parallel fetch mainly focuses on how to fetch, it is preferable to achieve both how and what to fetch. Lookahead OoO parallel fetch provides a viable means to achieve both important aspects of i-fetch.

VI. Lookahead Branch Prediction (LBP) for Parallel I-fetch in the BBC Technique

In the BBC-based i-fetch paradigm, the BPU can always fetch multiple flow-control instructions of basic blocks regardless of their programmed order. This allows the BPU cycles to overlap with i-fetch cycles and the cycles of other stages, including i-execution, which is not possible in sequential or parallel CPUs. To elaborate, the proposed i-fetch paradigm fetches multiple flow-control instructions in advance to initiate the fetching of contiguous instruction blocks from their associated basic blocks or fragments. Therefore, even if the branch prediction process takes multiple cycles, it can still produce a prediction result before all blocks are completely fetched, as long as the prediction latency is shorter than the number of blocks. This allows for more efficient overlapping of cycles between the BPU and other stages, such as i-fetch and i-execution, which is not possible with traditional sequential or parallel CPUs. The current practice requires cascading, overriding, or special branch predictors to hide multi-cycle branch prediction latencies. However, in the BBC-based i-fetch paradigm, the multi-cycle latency of the BPU is effectively hidden by concurrently fetching multiple instructions from two different decoupled subprograms in a lookahead manner. Fetching blocks of contiguous instructions from multiple threads is

one of the challenging tasks for designing a successful parallel fetch mechanism. To fetch from multiple locations, multiple near-future addresses in the dynamic control flow must be known by multiple program counters. Existing parallel fetch mechanisms, such as those introduced in references [1], have limitations due to the multiple replicated hardware components, including multiple program counters, sequencers, and buffers. In contrast, the BBC-based paradigm effectively handles parallel fetches from the dynamic control flow of the program with a simpler architecture and hardware components. The proposed fetch only requires a single program counter without any sequencer and multiple dedicated buffers. The architecture is almost like a sequential i-fetch one.

VII. LBP for Efficient BPU Access

The out-of-order parallel fetch in our paradigm operates at the basic block level and fetches only flow-control instructions, which need to be predicted. This significantly enhances i-fetch bandwidth and energy efficiency while providing alternatives to the limitations of existing fetch paradigms. In our i-fetch paradigm, only the necessary flow-control instructions are sequentially or in parallel fetched to the BPU for prediction before fetching contiguous instructions of basic blocks to the instruction queue. This lookahead fetch of flow-control instructions is predicted by the BPU, and then reordered to determine the branch behavior by the backend of the CPU. The multiple stages of i-fetch in OoO CPUs, such as ARM's Cortex-A72/-A57/-A15, can be a single fetch stage without pre-decoding the fetched instructions. The lookahead fetch cycles of branches to the BPU overlap with the fetches of contiguous instructions, hiding the latencies of taken branch prediction.

VIII. Dynamic Control Flow Transition with LBP

The parallel lookahead branch prediction fetches instructions seamlessly according to the dynamic control flow of the program. For example, up to three branches from consecutive basic blocks are fetched to a BPU, which processes them one per cycle in order. The remaining branches are discarded, and the next three branches from the predicted location are fetched in parallel. As a result, the contiguous instructions of the first basic block are fetched immediately, and the three branches of the control-flow subprogram are fetched at the same cycle. This prevents unnecessary instruction fetching from wrong paths due to taken branches in the dynamic control flow. In contrast, conventional parallel i-fetch mechanisms must fetch all instructions from three basic blocks, regardless of the predicted taken branch from the first basic block, and discard the instructions fetched from the second and third basic blocks. Furthermore, this lookahead dynamic control flow detection prevents i-cache pollutions during i-prefetch and i-fetch.

IX. Parallel I-Fetch from Two Decoupled Subprograms

Parallel instruction fetching allows for better tolerance of latency during instruction fetching in Out-of-Order CPUs. In this approach, blocks of contiguous instructions from the control-flow subprogram are fetched in parallel, which requires fewer cycles and less storage than fetching all instructions from multiple blocks of the functional subprogram. This allows for cache misses to be serviced multiple cycles early. Additionally, the proposed fetch paradigm is simple and low-power, with no prediction for prefetching flow-control instructions and expandable primary i-caches for upper/lower levels. This basic-block-level fetch paradigm can be applied for finer or coarser granularity fetch mechanisms and for in-order CPUs.

X. Low-power Loop Operations Efficiency

We designed small and low-power i-caches to reduce energy consumption in our fetch paradigm. However, smaller caches are more prone to cache misses, which can lead to performance degradation. We found that a small L1 cache (8KB) resulted in a 6% performance loss compared to a 128KB cache. To address this, we implemented continuous fetch and execution beyond a cache miss and overlapping multiple cache misses at the same cycles. Our decoupled subprograms also showed better resilience to cache misses due to their smaller size. Furthermore, our L2 caches are smaller and faster than traditional L2 caches, which contribute to resilience against cache misses and related stalls.

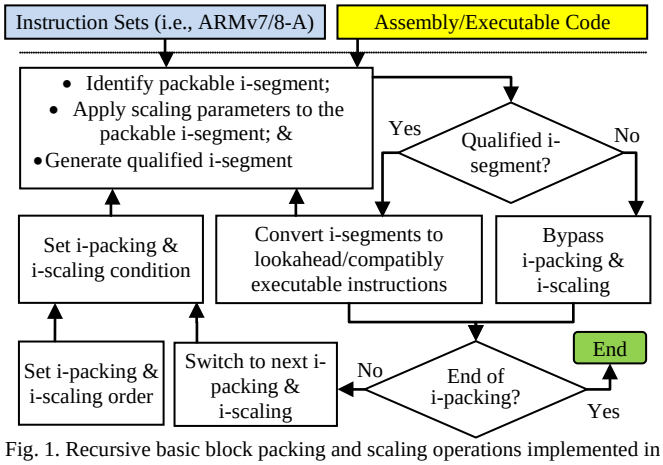

Fig. 1. Recursive basic block packing and scaling operations implemented in the lookahead code compilation in BBC

XI. Lookahead Code Compilation (LCC) in BBC

The lookahead code compilation in BBC utilizes parallel instruction fetching to efficiently fetch blocks of contiguous instructions from the control-flow subprogram. By fetching multiple instructions at once, latency is reduced and overall performance is improved. This technique optimizes cache resources, resulting in fewer cache misses and faster instruction execution. The parallel instruction fetching paradigm is a simple and low-power solution, suitable for fine or coarse-grained fetch mechanisms and in-order CPUs, including microcontrollers. Expandable primary i-caches for upper/lower levels improve performance and prefetching flow-control instructions is not required, simplifying the design and reducing power consumption. LLC in BBC effectively improves Out-of-Order CPU performance and energy efficiency while guaranteeing code compatibility. We evaluated code compatibility using lcc, measured the number of instructions fetched without being disrupted by a flow control instruction, and performed a dynamic i-stream extension experiment where the average number of instructions found in the extended i-stream was 2.45x greater than that in an i-stream compiled by gcc4.7.3. Without BBC, the average number of instructions was 15.31 instructions per i-stream.

XII. Conclusions and Future Directions

The LLC in BBC proposes the use of a replicated fetch unit for trace-granularity sequencing [2] to achieve high-performance processors. Multiple sequencers may be a more resilient solution to i-cache miss rates, but further research is needed to assess their effectiveness in light of technology trends. The LLC team plans to address the issue of waiting for earlier instructions to be fetched by renaming instructions out of order and issuing them to execution units.

References

[1] C. Kao, I. et al, “An embedded multi-resolution AMBA trace analyzer for microprocessor-based SoC,” IEEE DAC '07, pp. 477-482, 2007.

[2] B Grayson, et al., “Evolution of the Samsung Exynos CPU Microarchitecture,” ACM/IEEE 47th ISCA, pp. 40-51, 2020.