UNIVERSITY OF CAMBRIDGE
Department of Computer Science and Technology

The future of instruction-level parallelism (ILP)

Alexandra W. Chadwick, Márton Erdős, Utpal Bora, Akshay Bhosale, Yuxin Guo, Timothy M. Jones

Instruction-level parallelism

Programs consist of instructions. These are simple operations that a microprocessor can execute, such as addition or multiplication. Abstractly, executing a program consists of executing its instructions in program order. However, for decades, processors have utilised *instruction-level parallelism* (ILP) to execute multiple instructions simultaneously, and thus to execute programs faster than the implications of program order allow. Processor vendors have recently started to use ILP more aggressively to achieve performance.

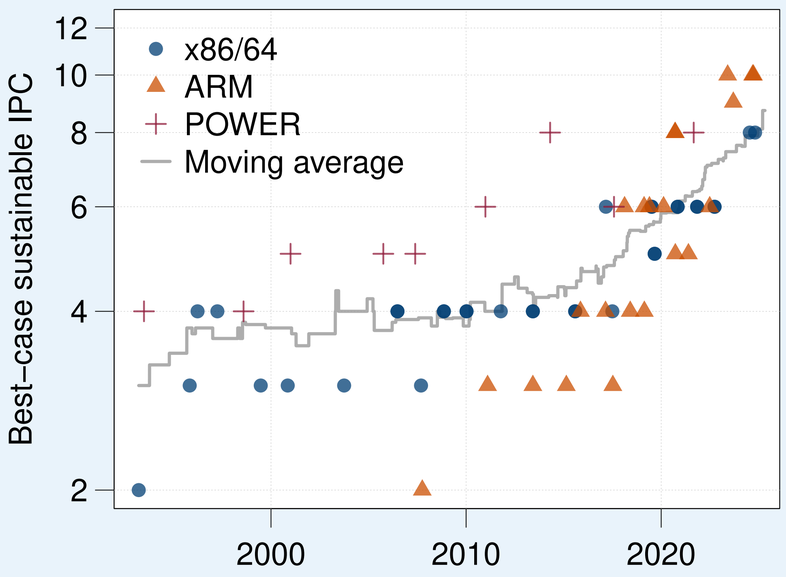


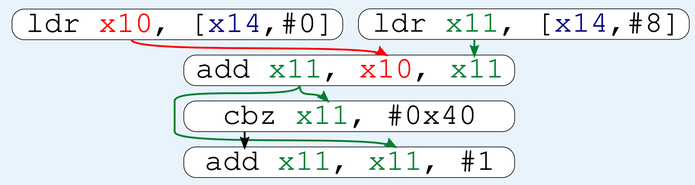
Figure 1: Maximum measurable instructions per cycle (IPC) of an optimal program executed on commercial processors from six vendors. Notice the increasing trend since 2015.

Dynamic dependency graphs (DDG)

When a program executes, each dynamic instruction may consume values produced by previous instructions, and produce values that future instructions may consume. By tracking these producer/consumer relationships we can build a *dynamic dependency graph* (DDG). This consists of one node per dynamic instruction, with directed edges from producers to consumers. The background of this poster is a tiny fraction of a DDG for the `deepsjeng` benchmark from the SPEC CPU2017 benchmark suite, compiled for AArch64. The full DDG for this program would have 2 trillion nodes, a modern high-performance processor would execute all of the instructions on this poster in about 0.3 microseconds.

```
ldr x10, [x14,#0]
ldr x11, [x14,#8]
add x11, x10, x11
cbz x11, #0x40
add x11, x11, #1
```

Program Order



Dynamic Dependency Graph

The longest path through a DDG is called the critical path. Assuming no processor can ever execute any instruction in less than one cycle, the length of the critical path should be a lower bound for the execution time of the program. This is because the instructions on the critical path must all execute, in sequence, in order for the program to finish. The critical path for the DDG in the background of the poster is shown in bold in the middle. The function that each instruction belongs to is indicated by a coloured background.

Speculative execution

High-performance microprocessors use a technique called speculative execution to improve performance. The processor predicts the result or outcome of some instructions, and can thus execute instructions that depend upon that result earlier. If the prediction is correct, this can allow a processor to achieve higher performance than the critical path in the DDG would suggest. Speculative execution is used for all branch outcomes in state-of-the-art processors, as well as to disambiguate memory operations.

We can nevertheless model the effects of speculative execution in a DDG; we simply delete any edges representing results that predictors would allow the processor to predict. In practice this means we delete the edges outward from most branch instructions as can be seen in the indicated positions. These branch instructions have no dependencies because a simulated branch predictor could predict their target.

Limits of ILP

Using the DDG, we can evaluate the limits of instruction-level parallelism in current code. We created the DDG for the first billion instructions of each of the SPEC CPU2017 benchmark programs. We break any dependencies that state-of-the-art branch prediction could successfully predict, thus modelling speculative execution. We then split the DDG into contiguous 'windows' of power-of-two sized groups of instructions. This models the execution paradigm of real-world processors, which cannot view the entire DDG simultaneously, but instead see only small finite groups of instructions (about $2^9 = 512$ for the biggest processors today).

For each of these windows we can use the DDG to compute the length of the critical path. This gives the minimum execution time of that window. If we divide the number of instructions in each window by its execution time, we obtain the average number of instructions that can execute in parallel in that part of the program. These average parallelism values vary considerably in different parts of the program.

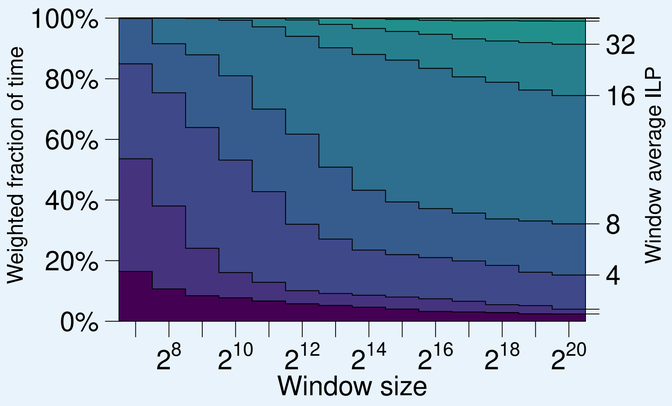


Figure 2: Worryingly, according to our model, even with extreme window scaling, a large majority of program execution time is spent in regions with an average ILP below 16.

Opportunities for the future?

The limits of ILP found in this study are concerning, suggesting that future microprocessors would struggle to ever achieve more than 16 IPC on average as shown in figure 2. This is concerning given the current scaling trend towards increasing hardware IPC resources shown in figure 1.

Can we use our knowledge of the DDG to spot improvements? Take a look at the indicated parts of the critical path in the DDG. Instructions are coloured according to the function they are part of in the source code. The calling convention utilised by this architecture dictates that callee functions must save and restore certain registers (x19–x29). We see sequences of instructions on the critical path that are enforcing this convention. The caller moves values it wishes to save into these registers, and the callee saves and restores them. The fact that these convention-enforcing instructions are on the critical path suggests a missed opportunity.

To investigate, we cut the edges in the DDG that just amounted to saving and restoring callee-saved registers, and recomputed the critical path. Doing so allows us to evaluate the potential IPC improvement for a processor that could somehow avoid the cost of this calling convention (either via architectural or microarchitectural innovation).

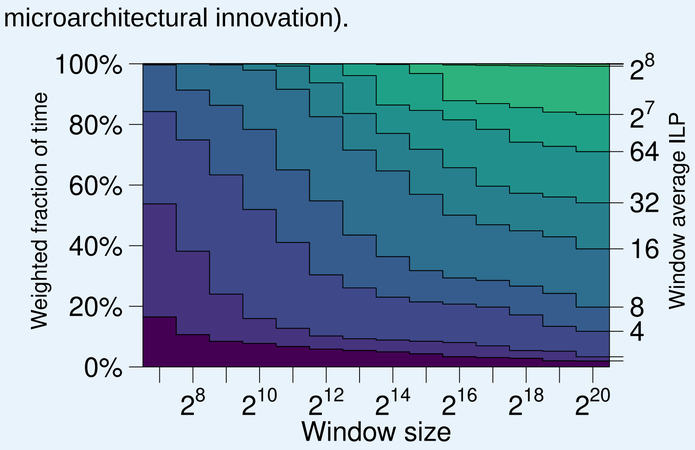


Figure 3: As figure 2, but with edges cut representing the calling convention. A significant boost to ILP is observed.

March 2025 — Alexandra.Chadwick@cl.cam.ac.uk
This work was supported by EPSRC (grant EP/W00576X/1) and Arm.
Computer Architecture Group
https://www.cst.cam.ac.uk/research/comparch