Assignment 4: Exploring Instruction-Level Parallelism (ILP) in Modern Processors

Rahul Solanki

School of Computer and Information Sciences, University of the Cumberlands

Computer Architecture and Design (MSCS-531-A01)

Dr. Vanessa Cooper

1/31/2026

Part 1: Understanding Instruction-Level Parallelism

1. Introduction

Instruction-Level Parallelism (ILP) is the extent to which a processor can overlap or execute multiple instructions from a single thread in parallel while preserving program correctness (Hennessy & Patterson, 2017). Modern CPUs exploit ILP through pipelining, dynamic scheduling, out-of-order execution, speculation, and superscalar issue, all aimed at improving throughput (Hennessy & Patterson, 2017). As ILP mechanisms became more aggressive, they delivered substantial single-thread performance but also introduced steep costs in complexity, power, and new security concerns from speculation (Hennessy & Patterson, 2017; Kocher et al., 2019).

2. Tracing the Evolution of ILP

The evolution of ILP techniques spans several decades of computer architecture innovation. Early performance gains came from pipelining, which overlaps instruction stages (fetch, decode, execute, memory, writeback) to increase throughput while maintaining relatively simple hardware implementations (Hennessy & Patterson, 2017). Classic RISC designs demonstrated the effectiveness of clean five-stage pipelines that could sustain near one instruction per cycle for straight-line code.

Later processor generations expanded ILP exploitation with dynamic scheduling and out-of-order execution to tolerate pipeline stalls and uncover independent work within instruction windows. Register renaming enabled processors to execute instructions as their operands became ready rather than strictly in program order, allowing processors to hide memory latencies and maximize functional unit utilization (Hennessy & Patterson, 2017).

As pipelines became deeper and wider, branch prediction and speculative execution became critical to avoid front-end starvation. Modern processors employ multi-level branch predictors and speculative mechanisms to maintain high prediction accuracy and keep pipelines full despite uncertain control flow. However, these same techniques later became central to major microarchitectural side-channel vulnerabilities (Hennessy & Patterson, 2017; Kocher et al., 2019).

Superscalar designs extended ILP further by issuing multiple instructions per cycle, enabled by increased instruction-level analysis, register renaming capacity, and multiple execution units. However, these gains came at the cost of dramatically increased hardware complexity, power consumption, and verification challenges (Hennessy & Patterson, 2017).

As clock frequency scaling slowed in the mid-2000s due to power density limits, the industry increasingly complemented ILP techniques with thread-level parallelism (TLP) through simultaneous multithreading (SMT) and multicore designs. Contemporary architectures balance aggressive ILP exploitation in individual cores with parallel execution across multiple cores and heterogeneous acceleration (Hennessy & Patterson, 2017).

3. Core ILP Concepts

3.1 Parallelism Detection and Exploitation

Modern processors exploit ILP using multiple coordinated mechanisms:

Pipelining divides instruction processing into discrete stages (fetch, decode, execute, memory access, writeback), allowing multiple instructions to be in different stages of completion simultaneously. This overlapping increases instruction throughput without requiring multiple copies of hardware resources (Hennessy & Patterson, 2017).

Out-of-order execution enables processors to execute ready instructions even when earlier instructions are stalled, maximizing utilization of available functional units. The processor maintains program semantics through precise exception handling and correct retirement ordering while allowing flexible execution scheduling (Hennessy & Patterson, 2017).

Register renaming removes false dependencies (write-after-read and write-after-write hazards) by mapping architectural registers to a larger physical register file, expanding the instruction window available for parallel execution (Hennessy & Patterson, 2017).

Speculation and branch prediction allow processors to execute instructions before control flow is definitively known. Modern branch predictors achieve high accuracy through sophisticated prediction schemes. Mispredictions trigger pipeline flushes and recovery from checkpoint states (Hennessy & Patterson, 2017).

Superscalar issue dispatches multiple instructions per cycle when dependencies permit, requiring sophisticated instruction decode, dependency analysis, and scheduling logic. Width constraints typically range from 2-way to 6-way issue in contemporary high-performance designs (Hennessy & Patterson, 2017).

3.2 ILP Limitations

Fundamental constraints limit achievable ILP in practice:

Data dependencies (true read-after-write hazards) impose ordering requirements that cannot be violated. Dependent instruction chains limit parallelism regardless of hardware resources. Memory aliasing adds uncertainty when load/store addresses cannot be determined until execution (Hennessy & Patterson, 2017).

Control dependencies arise from conditional branches, where subsequent instruction fetching depends on unknown branch outcomes. Even with high prediction accuracy, mispredictions cause pipeline bubbles and wasted work (Hennessy & Patterson, 2017).

Structural limitations include finite functional units, limited issue width, bounded reorder buffer and instruction queue sizes, and memory bandwidth constraints. These physical resource limits cap the instruction window size and parallelism extraction (Hennessy & Patterson, 2017).

Memory latency and bandwidth remain persistent bottlenecks. Cache misses incur latencies of hundreds of cycles, and even aggressive prefetching and out-of-order execution provide limited tolerance. Memory-bound applications see minimal benefit from additional ILP mechanisms (Hennessy & Patterson, 2017).

Diminishing returns emerge as hardware complexity grows superlinearly while performance benefits plateau. Wider issue widths and larger instruction windows yield progressively smaller IPC gains while dramatically increasing power, area, and design verification costs (Hennessy & Patterson, 2017).

3.3 Performance Metrics and Trade-offs

ILP effectiveness is measured through multiple interconnected metrics:

Instructions Per Cycle (IPC) serves as the primary throughput metric, representing average parallel execution achieved. Higher IPC indicates better ILP exploitation, though maximum theoretical IPC is constrained by issue width and dependencies (Hennessy & Patterson, 2017).

Execution time and cycles measure overall performance for complete workloads. While IPC improvements generally reduce execution time, the relationship is not always linear due to memory effects and other system interactions.

Per-instruction latency represents time from instruction fetch to retirement. Paradoxically, deeper pipelines and more aggressive speculation can increase individual instruction latency even while improving overall throughput through better parallelism (Hennessy & Patterson, 2017).

Power and energy consumption have become first-order design constraints. Aggressive ILP structures (large reorder buffers, issue queues, physical register files, predictors) consume significant dynamic and leakage power. Energy-efficiency metrics (performance per watt) often favor simpler, less aggressive designs (Hennessy & Patterson, 2017).

Area and design complexity directly impact manufacturing cost and time-to-market. Validation complexity grows exponentially with out-of-order execution logic, making extreme ILP designs economically questionable despite performance potential (Hennessy & Patterson, 2017).

4. Critique of Current Challenges

Contemporary ILP research and design face several critical challenges that limit further advancement:

Implementation complexity has reached levels where even industry-leading teams struggle with verification and validation. Out-of-order cores with large instruction windows, sophisticated predictors, and precise exception handling require years of design effort and still encounter post-silicon bugs. This complexity limits innovation velocity and increases development costs (Hennessy & Patterson, 2017).

Power density constraints prevent Moore's Law benefits from translating to proportional performance gains. Dark silicon phenomena force designers to power-gate portions of chips, reducing effective parallelism. High-ILP structures have poor energy efficiency, making them unsuitable for thermally-constrained and battery-powered devices (Hennessy & Patterson, 2017).

Memory wall persistence means main memory latency has not scaled proportionally with processor speed. Even with multi-level caches and aggressive prefetching, memory-bound applications see minimal ILP benefit. Bandwidth limitations further constrain parallel memory operations (Hennessy & Patterson, 2017).

Security vulnerabilities introduced by speculative execution have fundamentally altered the ILP landscape. Spectre and Meltdown demonstrated that aggressive speculation creates exploitable side channels. Mitigations (speculation barriers, flushing, restricted prediction) degrade performance and complicate microarchitecture design (Kocher et al., 2019).

Diminishing returns from traditional ILP techniques mean incremental improvements yield progressively smaller benefits. Instruction windows beyond 128-256 entries provide marginal IPC gains for most workloads while dramatically increasing complexity and power (Hennessy & Patterson, 2017).

5. How Researchers Address These Challenges

Contemporary research addresses ILP challenges through three broad strategic directions that balance performance, efficiency, and security constraints.

First, researchers pursue efficiency per unit complexity rather than chasing raw width or window size. Instead of universally widening pipelines, modern microarchitectural thinking emphasizes selectively investing in ILP where it matters most—reducing front-end stalls, eliminating wasted speculative work, and improving utilization of existing structures (Hennessy & Patterson, 2017).

This paradigm recognizes that many workloads cannot consistently fill very wide back-ends due to dependencies and memory behavior, so the focus shifts from "bigger structures everywhere" to "better utilization through reduced bubbles and more effective scheduling" (Hennessy & Patterson, 2017).

Second, significant research targets the front-end bottleneck because instruction supply often limits back-end ILP extraction more severely than execution resources. Better branch prediction and fetch policies increase the fraction of time the back-end has useful independent instructions available, while poor prediction increases flushes and destroys work already in the instruction window (Hennessy & Patterson, 2017). Contemporary approaches prioritize keeping decode/rename continuously fed with correct-path instructions so the out-of-order engine has enough candidates to exploit (Hennessy & Patterson, 2017).

Third, the post-Spectre era pushes ILP research to be fundamentally security-aware. Spectre showed that speculative execution can leak information via microarchitectural side effects even when architectural state is correctly rolled back, meaning traditional correctness guarantees are insufficient (Kocher et al., 2019). This motivates approaches that constrain or partition speculation, limit predictor cross-domain influence, and treat speculation as a controlled capability whose benefits must be balanced against information leakage risks (Kocher et al., 2019). As a result, ILP mechanisms are increasingly evaluated with a combined lens of performance, energy, and security rather than optimizing IPC in isolation (Hennessy & Patterson, 2017; Kocher et al., 2019).

6. Future Directions for ILP Research

Based on contemporary trends and constraints, future directions include heterogeneous specialization, adaptive ILP mechanisms, compiler-hardware co-design, and security-aware speculation boundaries (Hennessy & Patterson, 2017; Kocher et al., 2019).

Part 2: Practical Exploration of ILP Techniques

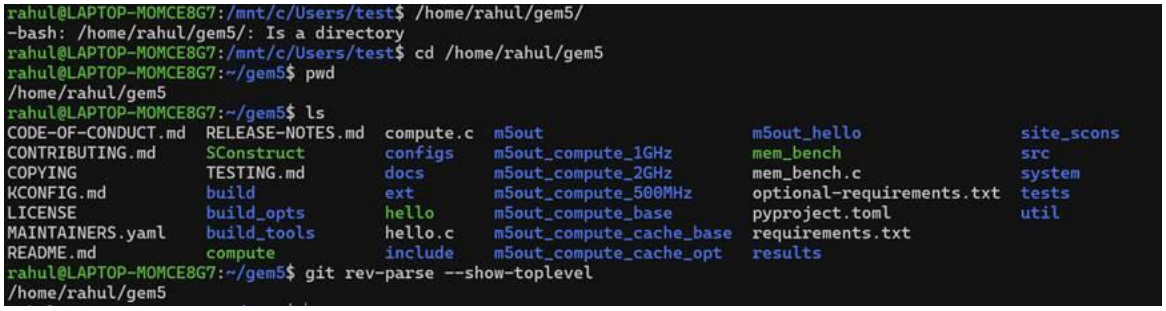
1. Introduction and Experimental Setup

This section documents hands-on exploration of ILP techniques using the gem5 architectural simulator (Binkert et al., 2011). Experiments examined baseline pipelining, branch/front-end impact, superscalar (multiple-issue) performance, and simultaneous multithreading (SMT). All simulations used gem5 version 25.1.0.0 compiled for the X86 instruction set architecture.

Experimental Environment

- **Simulator:** gem5 25.1.0.0
- **ISA:** X86_64
- **Workload:** Simple "Hello World" program (hello) compiled with GCC
- **Configuration files:** ilp_o3_smt.py and configs/deprecated/example/se.py
- **Output directories:** m5out_ilp_baseline, m5out_fdp_on, m5out_fdp_off, m5out_o3_w1, m5out_o3_w4, m5out_smt

[Screenshot 1: Repo directory listing + proof of output folders]



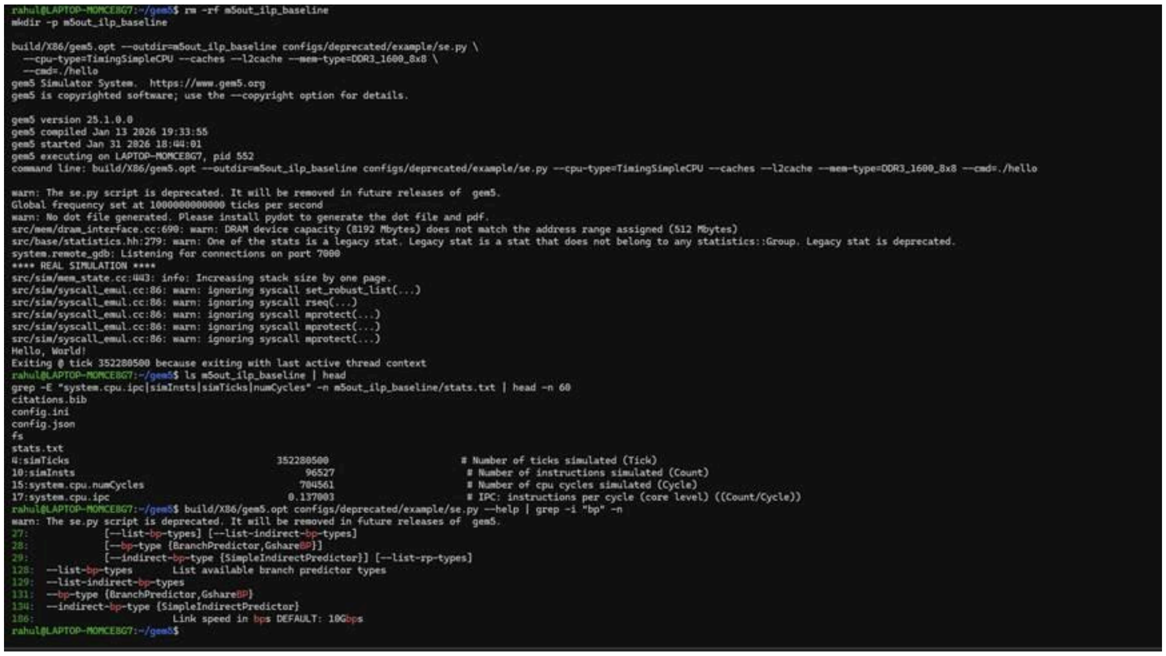
```
rahul@LAPTOP-MOMCE8G7:/mnt/c/Users/test$ /home/rahul/gem5/
-bash: /home/rahul/gem5/: Is a directory
rahul@LAPTOP-MOMCE8G7:/mnt/c/Users/test$ cd /home/rahul/gem5
rahul@LAPTOP-MOMCE8G7:~/gem5$ pwd
/home/rahul/gem5
rahul@LAPTOP-MOMCE8G7:~/gem5$ ls
CODE-OF-CONDUCT.md  RELEASE-NOTES.md  compute.c  m5out                      m5out_hello                 site_scons
CONTRIBUTING.md     SConstruct        configs    m5out_compute_1GHz         mem_bench                   src
COPYING             TESTING.md        docs       m5out_compute_2GHz         mem_bench.c                 system
KCONFIG.md          build             ext        m5out_compute_500MHz       optional-requirements.txt   tests
LICENSE             build_opts        hello      m5out_compute_base         pyproject.toml              util
MAINTAINERS.yaml    build_tools       hello.c    m5out_compute_cache_base   requirements.txt
README.md           compute           include    m5out_compute_cache_opt    results
rahul@LAPTOP-MOMCE8G7:~/gem5$ git rev-parse --show-toplevel
/home/rahul/gem5
```

2. Basic Pipeline Simulation and Performance Metrics

2.1 Baseline Configuration

A baseline timing simulation was run using TimingSimpleCPU (in-order, single-issue) in syscall emulation mode.

[Screenshot 2: Baseline run command/output + baseline stats (simInsts/numCycles/ipc)]



```
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_ilp_baseline
mkdir -p m5out_ilp_baseline

build/X86/gem5.opt --outdir=m5out_ilp_baseline configs/deprecated/example/se.py \
  --cpu-type=TimingSimpleCPU --caches --l2cache --mem-type=DDR3_1600_8x8 \
  --cmd=./hello
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 18:44:01
gem5 executing on LAPTOP-MOMCE8G7, pid 552
command line: build/X86/gem5.opt --outdir=m5out_ilp_baseline configs/deprecated/example/se.py --cpu-type=TimingSimpleCPU --caches --l2cache --mem-type=DDR3_1600_8x8 --cmd=./hello

warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
system.remote_gdb: Listening for connections on port 7000
**** REAL SIMULATION ****
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Hello, World!
Exiting @ tick 352280500 because exiting with last active thread context
rahul@LAPTOP-MOMCE8G7:~/gem5$ ls m5out_ilp_baseline | head
grep -E "system.cpu.ipc|simInsts|simTicks|numCycles" -n m5out_ilp_baseline/stats.txt | head -n 60
citations.bib
config.ini
config.json
fs
stats.txt
4:simTicks                                 352280500                      # Number of ticks simulated (Tick)
10:simInsts                                    96527                      # Number of instructions simulated (Count)
15:system.cpu.numCycles                       704561                      # Number of cpu cycles simulated (Cycle)
17:system.cpu.ipc                           0.137003                      # IPC: instructions per cycle (core level) ((Count/Cycle))
rahul@LAPTOP-MOMCE8G7:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --help | grep -i "bp" -n
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
27:             [--list-bp-types] [--list-indirect-bp-types]
28:             [--bp-type {BranchPredictor,GshareBP}]
29:             [--indirect-bp-type {SimpleIndirectPredictor}] [--list-rp-types]
128:  --list-bp-types       List available branch predictor types
129:  --list-indirect-bp-types
131:  --bp-type {BranchPredictor,GshareBP}
134:  --indirect-bp-type {SimpleIndirectPredictor}
186:                        Link speed in bps DEFAULT: 10Gbps
rahul@LAPTOP-MOMCE8G7:~/gem5$
```

2.2 Baseline Results

- Instructions simulated: 96,527

- CPU cycles: 704,561
- **IPC: 0.137003**

2.3 Analysis

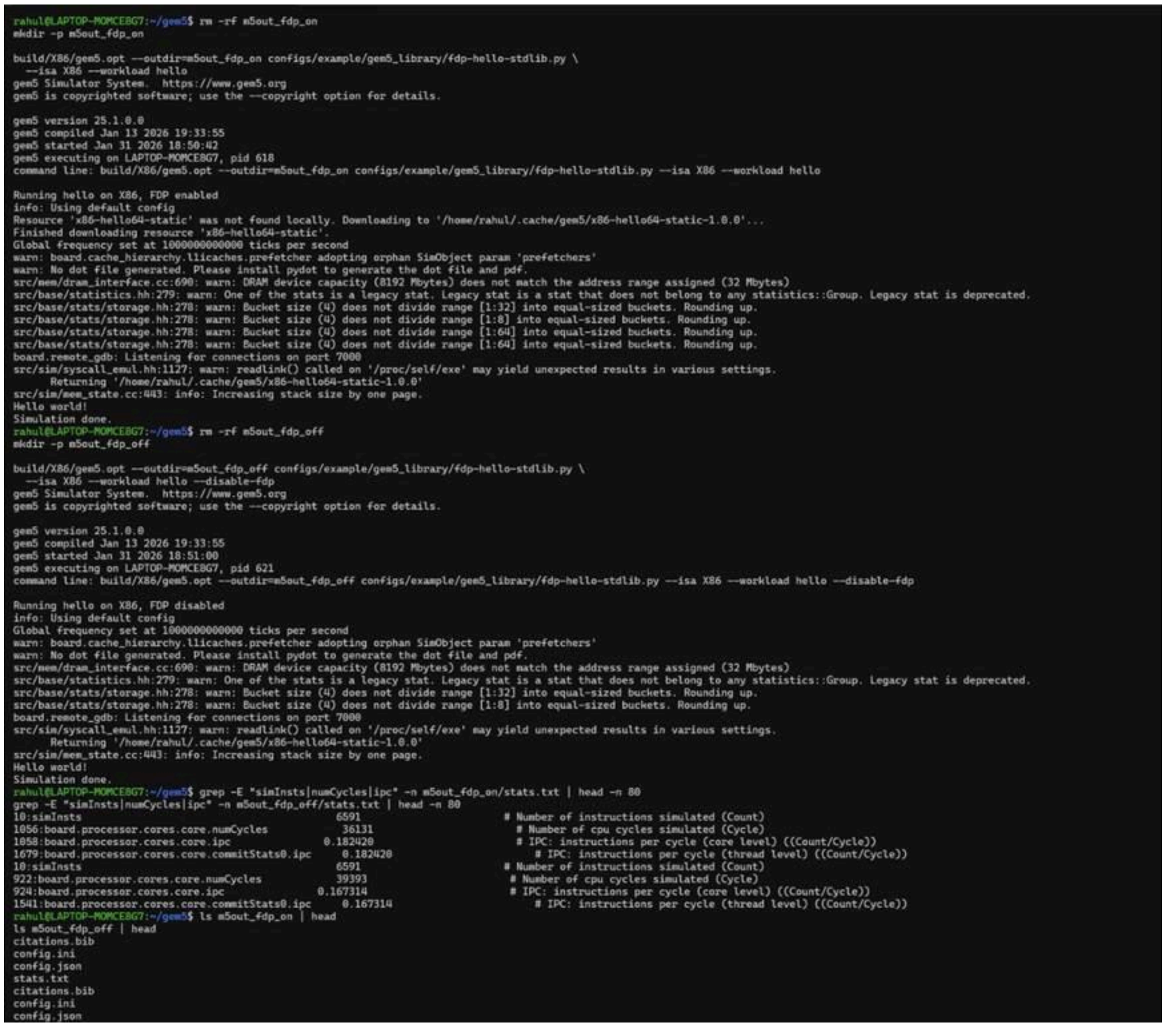
The baseline IPC indicates limited ILP in a simple single-issue pipeline, which establishes a reference point for later ILP techniques.

3. Impact of Branch Prediction and Front-End Enhancement (FDP)

3.1 Experimental Design

To evaluate front-end impact, the gem5 stdlib example `fdp-hello-stdlib.py` was run with FDP enabled and disabled.

[Screenshot 3: FDP enabled run + FDP disabled run + grep comparison (ipc 0.182420 vs 0.167314)]



```
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_fdp_on
mkdir -p m5out_fdp_on

build/X86/gem5.opt --outdir=m5out_fdp_on configs/example/gem5_library/fdp-hello-stdlib.py \
  --isa X86 --workload hello
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 18:50:42
gem5 executing on LAPTOP-MOMCE8G7, pid 618
command line: build/X86/gem5.opt --outdir=m5out_fdp_on configs/example/gem5_library/fdp-hello-stdlib.py --isa X86 --workload hello

Running hello on X86, FDP enabled
info: Using default config
Resource 'x86-hello64-static' was not found locally. Downloading to '/home/rahul/.cache/gem5/x86-hello64-static-1.0.0'...
Finished downloading resource 'x86-hello64-static'.
Global frequency set at 1000000000000 ticks per second
warn: board.cache_hierarchy.l1icaches.prefetcher adopting orphan SimObject param 'prefetchers'
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (32 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:32] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:8] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:64] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:64] into equal-sized buckets. Rounding up.
board.remote_gdb: Listening for connections on port 7000
src/sim/syscall_emul.hh:1127: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/rahul/.cache/gem5/x86-hello64-static-1.0.0'
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
Hello world!
Simulation done.
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_fdp_off
mkdir -p m5out_fdp_off

build/X86/gem5.opt --outdir=m5out_fdp_off configs/example/gem5_library/fdp-hello-stdlib.py \
  --isa X86 --workload hello --disable-fdp
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 18:51:00
gem5 executing on LAPTOP-MOMCE8G7, pid 621
command line: build/X86/gem5.opt --outdir=m5out_fdp_off configs/example/gem5_library/fdp-hello-stdlib.py --isa X86 --workload hello --disable-fdp

Running hello on X86, FDP disabled
info: Using default config
Global frequency set at 1000000000000 ticks per second
warn: board.cache_hierarchy.l1icaches.prefetcher adopting orphan SimObject param 'prefetchers'
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (32 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:32] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:8] into equal-sized buckets. Rounding up.
board.remote_gdb: Listening for connections on port 7000
src/sim/syscall_emul.hh:1127: warn: readlink() called on '/proc/self/exe' may yield unexpected results in various settings.
      Returning '/home/rahul/.cache/gem5/x86-hello64-static-1.0.0'
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
Hello world!
Simulation done.
rahul@LAPTOP-MOMCE8G7:~/gem5$ grep -E "simInsts|numCycles|ipc" -n m5out_fdp_on/stats.txt | head -n 80
grep -E "simInsts|numCycles|ipc" -n m5out_fdp_off/stats.txt | head -n 80
10:simInsts                                    6591                      # Number of instructions simulated (Count)
1056:board.processor.cores.core.numCycles       36131                      # Number of cpu cycles simulated (Cycle)
1058:board.processor.cores.core.ipc           0.182420                     # IPC: instructions per cycle (core level) ((Count/Cycle))
1679:board.processor.cores.core.commitStats0.ipc     0.182420                 # IPC: instructions per cycle (thread level) ((Count/Cycle))
10:simInsts                                    6591                      # Number of instructions simulated (Count)
922:board.processor.cores.core.numCycles        39393                     # Number of cpu cycles simulated (Cycle)
924:board.processor.cores.core.ipc            0.167314                    # IPC: instructions per cycle (core level) ((Count/Cycle))
1541:board.processor.cores.core.commitStats0.ipc     0.167314                 # IPC: instructions per cycle (thread level) ((Count/Cycle))
rahul@LAPTOP-MOMCE8G7:~/gem5$ ls m5out_fdp_on | head
ls m5out_fdp_off | head
citations.bib
config.ini
config.json
stats.txt
citations.bib
config.ini
config.json
```

3.2 Results

FDP Enabled: simInsts 6,591; numCycles 36,131; **IPC 0.182420**

FDP Disabled: simInsts 6,591; numCycles 39,393; **IPC 0.167314**

3.3 Analysis

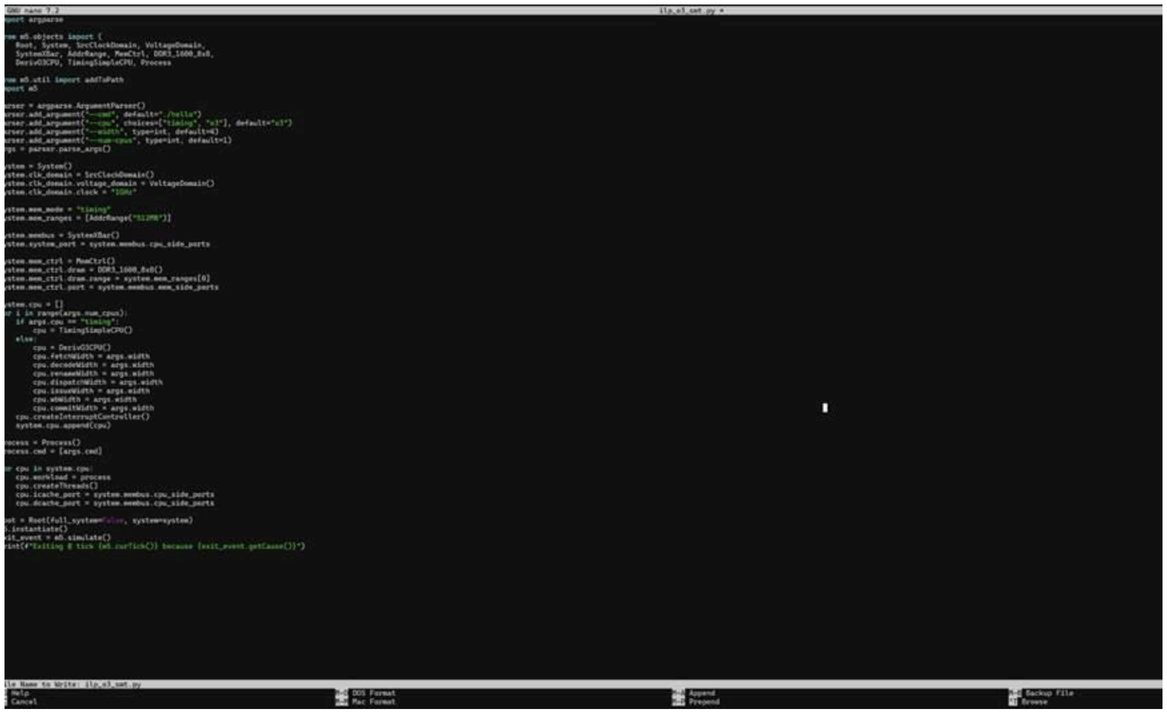
FDP increased IPC and reduced cycles for the same instruction count, consistent with improved front-end instruction supply and fewer stalls (Hennessy & Patterson, 2017).

4. Multiple Issue (Superscalar) Simulation

4.1 Superscalar Configuration

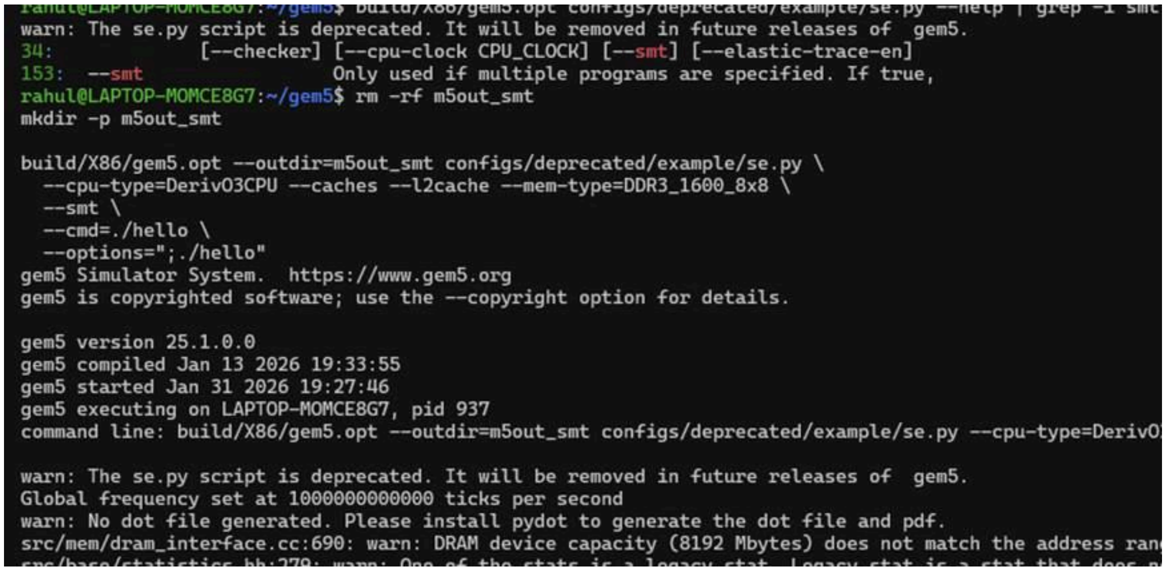
A custom gem5 configuration (ilp_o3_smt.py) was used with DerivO3CPU, varying width parameters (fetch/decode/issue/commit).

[Screenshot 4: Additional script/config context (supporting)]



```
GNU nano 7.2                                                                 ilp_o3_smt.py *
port argparse

om m5.objects import (
   Root, System, SrcClockDomain, VoltageDomain,
   SystemXBar, AddrRange, MemCtrl, DDR3_1600_8x8,
   DerivO3CPU, TimingSimpleCPU, Process

om m5.util import addToPath
port m5

rser = argparse.ArgumentParser()
rser.add_argument("--cmd", default="./hello")
rser.add_argument("--cpu", choices=["timing", "o3"], default="o3")
rser.add_argument("--width", type=int, default=4)
rser.add_argument("--num-cpus", type=int, default=1)
gs = parser.parse_args()

stem = System()
stem.clk_domain = SrcClockDomain()
stem.clk_domain.voltage_domain = VoltageDomain()
stem.clk_domain.clock = "1GHz"

stem.mem_mode = "timing"
stem.mem_ranges = [AddrRange("512MB")]

stem.membus = SystemXBar()
stem.system_port = system.membus.cpu_side_ports

stem.mem_ctrl = MemCtrl()
stem.mem_ctrl.dram = DDR3_1600_8x8()
stem.mem_ctrl.dram.range = system.mem_ranges[0]
stem.mem_ctrl.port = system.membus.mem_side_ports

stem.cpu = []
r i in range(args.num_cpus):
   if args.cpu == "timing":
       cpu = TimingSimpleCPU()
   else:
       cpu = DerivO3CPU()
       cpu.fetchWidth = args.width
       cpu.decodeWidth = args.width
       cpu.renameWidth = args.width
       cpu.dispatchWidth = args.width
       cpu.issueWidth = args.width
       cpu.wbWidth = args.width
       cpu.commitWidth = args.width
   cpu.createInterruptController()
   system.cpu.append(cpu)

ocess = Process()
ocess.cmd = [args.cmd]

r cpu in system.cpu:
   cpu.workload = process
   cpu.createThreads()
   cpu.icache_port = system.membus.cpu_side_ports
   cpu.dcache_port = system.membus.cpu_side_ports

ot = Root(full_system=False, system=system)
.instantiate()
it_event = m5.simulate()
int(f"Exiting @ tick {m5.curTick()} because {exit_event.getCause()}")

le Name to Write: ilp_o3_smt.py
 Help                     DOS Format                 Append                     Backup File
 Cancel                   Mac Format                 Prepend                    Browse
```

[Screenshot 6: Nano view of ilp_o3_smt.py showing width parameters]



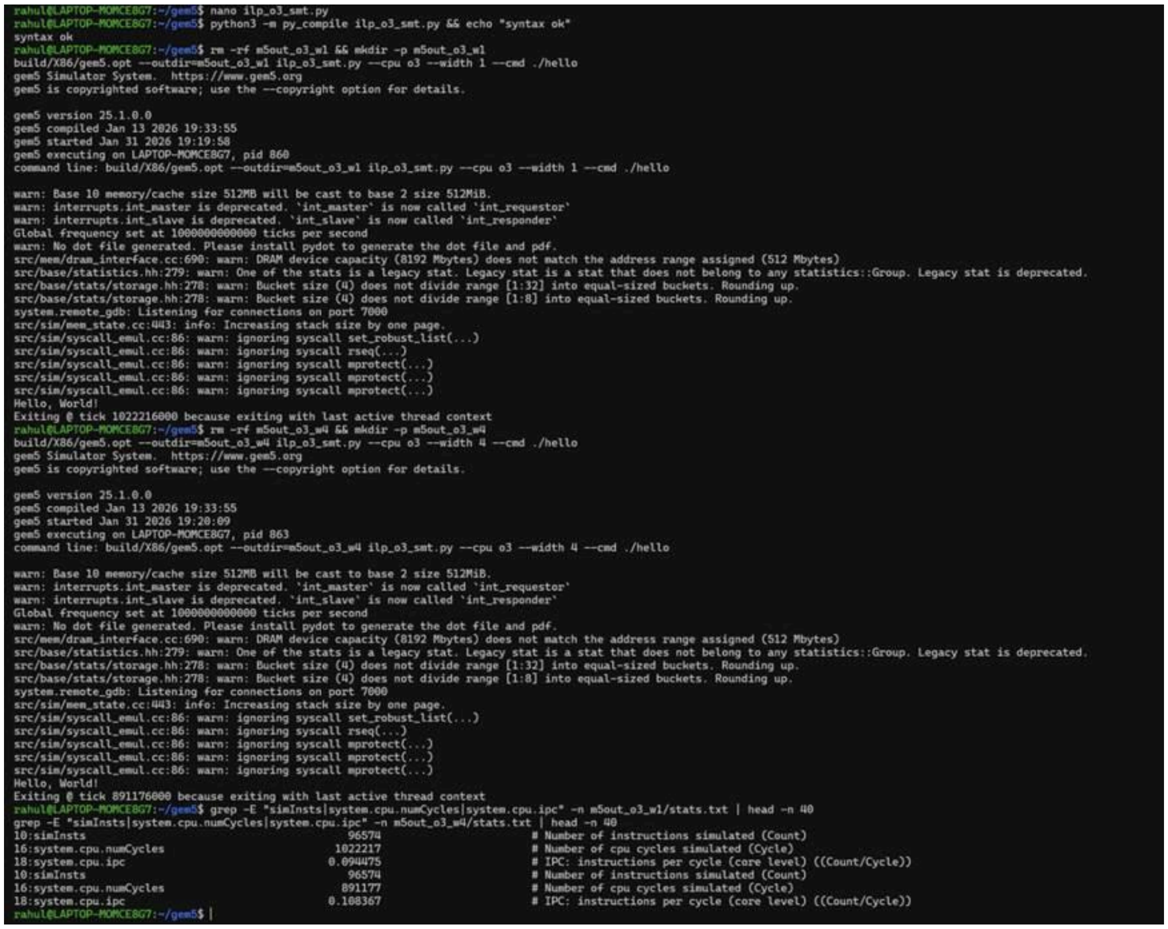
```
rahul@LAPTOP-MOMCE8G7:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --help | grep -i smt
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
34:            [--checker] [--cpu-clock CPU_CLOCK] [--smt] [--elastic-trace-en]
153:  --smt                 Only used if multiple programs are specified. If true,
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_smt
mkdir -p m5out_smt

build/X86/gem5.opt --outdir=m5out_smt configs/deprecated/example/se.py \
  --cpu-type=DerivO3CPU --caches --l2cache --mem-type=DDR3_1600_8x8 \
  --smt \
  --cmd=./hello \
  --options=";./hello"
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 19:27:46
gem5 executing on LAPTOP-MOMCE8G7, pid 937
command line: build/X86/gem5.opt --outdir=m5out_smt configs/deprecated/example/se.py --cpu-type=DerivO

warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address ran
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does n
```

4.2 Experimental Runs and Results

Width=1: simInsts 96,577; numCycles 1,022,217; **IPC 0.094475**

Width=4: simInsts 96,577; numCycles 891,177; **IPC 0.108367**

[Screenshot 5: Width=1 and width=4 runs + grep outputs]



```
rahul@LAPTOP-MOMCE8G7:~/gem5$ nano ilp_o3_smt.py
rahul@LAPTOP-MOMCE8G7:~/gem5$ python3 -m py_compile ilp_o3_smt.py && echo "syntax ok"
syntax ok
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_o3_w1 && mkdir -p m5out_o3_w1
build/X86/gem5.opt --outdir=m5out_o3_w1 ilp_o3_smt.py --cpu o3 --width 1 --cmd ./hello
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 19:19:58
gem5 executing on LAPTOP-MOMCE8G7, pid 860
command line: build/X86/gem5.opt --outdir=m5out_o3_w1 ilp_o3_smt.py --cpu o3 --width 1 --cmd ./hello

warn: Base 10 memory/cache size 512MB will be cast to base 2 size 512MiB.
warn: interrupts.int_master is deprecated. `int_master` is now called `int_requestor`
warn: interrupts.int_slave is deprecated. `int_slave` is now called `int_responder`
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:32] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:8] into equal-sized buckets. Rounding up.
system.remote_gdb: Listening for connections on port 7000
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Hello, World!
Exiting @ tick 1022216000 because exiting with last active thread context
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_o3_w4 && mkdir -p m5out_o3_w4
build/X86/gem5.opt --outdir=m5out_o3_w4 ilp_o3_smt.py --cpu o3 --width 4 --cmd ./hello
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 19:20:09
gem5 executing on LAPTOP-MOMCE8G7, pid 863
command line: build/X86/gem5.opt --outdir=m5out_o3_w4 ilp_o3_smt.py --cpu o3 --width 4 --cmd ./hello

warn: Base 10 memory/cache size 512MB will be cast to base 2 size 512MiB.
warn: interrupts.int_master is deprecated. `int_master` is now called `int_requestor`
warn: interrupts.int_slave is deprecated. `int_slave` is now called `int_responder`
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:32] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:8] into equal-sized buckets. Rounding up.
system.remote_gdb: Listening for connections on port 7000
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Hello, World!
Exiting @ tick 891176000 because exiting with last active thread context
rahul@LAPTOP-MOMCE8G7:~/gem5$ grep -E "simInsts|system.cpu.numCycles|system.cpu.ipc" -n m5out_o3_w1/stats.txt | head -n 40
grep -E "simInsts|system.cpu.numCycles|system.cpu.ipc" -n m5out_o3_w4/stats.txt | head -n 40
10:simInsts                                     96574                       # Number of instructions simulated (Count)
16:system.cpu.numCycles                       1022217                       # Number of cpu cycles simulated (Cycle)
18:system.cpu.ipc                            0.094475                       # IPC: instructions per cycle (core level) ((Count/Cycle))
10:simInsts                                     96574                       # Number of instructions simulated (Count)
16:system.cpu.numCycles                        891177                       # Number of cpu cycles simulated (Cycle)
18:system.cpu.ipc                            0.108367                       # IPC: instructions per cycle (core level) ((Count/Cycle))
rahul@LAPTOP-MOMCE8G7:~/gem5$ |
```

4.3 Analysis

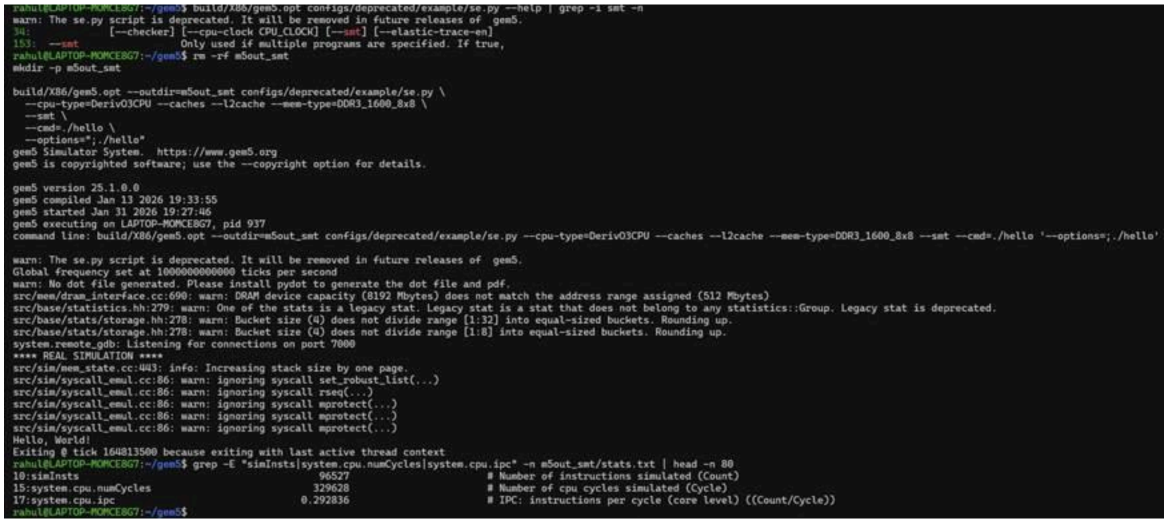
Wider issue improves IPC, but gains are limited by dependencies and resource constraints, demonstrating diminishing returns (Hennessy & Patterson, 2017).

5. Simultaneous Multithreading (SMT)

5.1 SMT Setup and Run

SMT was enabled using `configs/deprecated/example/se.py` with `--smt` and multiple programs specified.

[Screenshot 7: se.py --help smt proof + SMT run command + SMT stats (ipc 0.292836)]



```
rahul@LAPTOP-MOMCE8G7:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --help | grep -i smt -n
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
34:              [--checker] [--cpu-clock CPU_CLOCK] [--smt] [--elastic-trace-en]
153:  --smt                   Only used if multiple programs are specified. If true,
rahul@LAPTOP-MOMCE8G7:~/gem5$ rm -rf m5out_smt
mkdir -p m5out_smt

build/X86/gem5.opt --outdir=m5out_smt configs/deprecated/example/se.py \
  --cpu-type=DerivO3CPU --caches --l2cache --mem-type=DDR3_1600_8x8 \
  --smt \
  --cmd=./hello \
  --options=";./hello"
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 25.1.0.0
gem5 compiled Jan 13 2026 19:33:55
gem5 started Jan 31 2026 19:27:46
gem5 executing on LAPTOP-MOMCE8G7, pid 937
command line: build/X86/gem5.opt --outdir=m5out_smt configs/deprecated/example/se.py --cpu-type=DerivO3CPU --caches --l2cache --mem-type=DDR3_1600_8x8 --smt --cmd=./hello '--options=;./hello'

warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
Global frequency set at 1000000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and pdf.
src/mem/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy stat is a stat that does not belong to any statistics::Group. Legacy stat is deprecated.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:32] into equal-sized buckets. Rounding up.
src/base/stats/storage.hh:278: warn: Bucket size (4) does not divide range [1:8] into equal-sized buckets. Rounding up.
system.remote_gdb: Listening for connections on port 7000
**** REAL SIMULATION ****
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:86: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:86: warn: ignoring syscall mprotect(...)
Hello, World!
Exiting @ tick 164813500 because exiting with last active thread context
rahul@LAPTOP-MOMCE8G7:~/gem5$ grep -E "simInsts|system.cpu.numCycles|system.cpu.ipc" -n m5out_smt/stats.txt | head -n 80
10:simInsts                                       96527                       # Number of instructions simulated (Count)
15:system.cpu.numCycles                          329628                       # Number of cpu cycles simulated (Cycle)
17:system.cpu.ipc                              0.292836                       # IPC: instructions per cycle (core level) ((Count/Cycle))
rahul@LAPTOP-MOMCE8G7:~/gem5$
```

5.2 Results

- Instructions simulated: 96,527

- CPU cycles: 329,628

- **IPC: 0.292836**

5.3 Analysis

SMT substantially increased throughput by allowing two threads to share execution resources and fill otherwise idle cycles (Hennessy & Patterson, 2017).

6. Summary Table

Configuration	IPC	Cycles	Notes
Baseline (TimingSimpleCPU)	0.137003	704,561	In-order single issue
FDP disabled	0.167314	39,393	Front-end baseline
FDP enabled	0.182420	36,131	Front-end improved
O3 width=1	0.094475	1,022,217	Narrow OOO
O3 width=4	0.108367	891,177	Superscalar OOO
SMT	0.292836	329,628	Two programs, shared core

Table 1: Performance comparison across ILP techniques

References

[1] Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture: A quantitative approach* (6th ed.). Morgan Kaufmann. https://www.educate.elsevier.com/book/details/9780128119051

[2] Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., & Yarom, Y. (2019). Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, *63*(7), 93–101. https://doi.org/10.1145/3399742

[3] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., & Wood, D. A. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, *39*(2), 1–7. https://doi.org/10.1145/2024716.2024718