Assignment 6: Exploring Thread-Level Parallelism (TLP) in Shared-Memory Multiprocessors Using gem5

Milan Bista

University of Cumberlands

MSCS-531-M50: Computer Architecture and Design

Instructors: Machica Mcclain / Charles Lively

https://github.com/mbista25742/MSCS-531-M50-Assignment6

Introduction:

Thread-Level Parallelism (TLP) is critical for leveraging multi-core processors and shared-memory architectures in modern computing. This paper reviews the historical development of TLP, explores its core concepts, and critiques contemporary challenges such as concurrency bugs, scalability issues, and energy efficiency. A synthesis of recent research reveals promising trends in programming models, hardware enhancements, and machine learning for thread management. The findings suggest future directions for TLP, focusing on many-core architectures, integration with SIMD, and specialized hardware designs.

The shift from single core to multi-core processors has profoundly impacted computer architecture. Central to this evolution is Thread-Level Parallelism (TLP), which enables multiple threads to execute concurrently, improving throughput and efficiency. Shared-memory multiprocessor systems, combined with advances in hardware and programming models, have catalyzed this transformation. This paper critically reviews recent research to provide a comprehensive understanding of TLP, its challenges, and future directions.

1. Historical Development of TLP

TLP's trajectory reflects the broader evolution of computing. Early systems utilized time-sharing, while the advent of multi-core processors in the mid-2000s marked a paradigm shift. Intel's Core Duo (2006) exemplified this change, offering parallelism within a single chip. Programming models evolved from explicit threading models, like POSIX threads, to task-based systems, such as Intel’s Threading Building Blocks (TBB). Hardware innovations, including cache coherence protocols and NUMA architectures, have reduced latencies in shared-memory systems.

2.Core Concepts in TLP

TLP is governed by several foundational principles:

-Parallelism Models

Shared memory and message passing represent the two primary paradigms.

- Shared Memory: Threads access a common address space, minimizing data exchange overhead. However, it introduces challenges like cache coherence.
- Message Passing: Explicit message exchanges enable scalability but increase complexity in thread coordination.

-Synchronization and Communication

Effective synchronization is critical to reducing contention. Traditional approaches include:

- Locks and Mutexes for ensuring mutual exclusion.
- Lock-Free Algorithms that use atomic operations to improve performance.

Load Balancing and Scheduling

Dynamic scheduling techniques, such as work-stealing, allow threads to redistribute tasks dynamically, ensuring efficient utilization of resources.

Performance Metrics

TLP performance is measured through metrics like throughput, latency, and scalability. Trade-offs between these metrics often depend on the workload and system architecture.

3.Current Challenges in TLP

Despite advancements, several challenges persist:

Concurrency Bugs and Race Conditions

Concurrency bugs remain a significant barrier to adopting TLP. Tools like ThreadSanitizer can identify issues, but complete prevention requires improved abstractions.

Scalability and Amdahl's Law

Scalability is constrained by Amdahl's Law, which highlights the diminishing returns of parallelism for serial code segments. Algorithm redesign and fine-grained parallelism are necessary for addressing these limitations.

Heterogeneous Architectures

Integrating CPUs, GPUs, and specialized accelerators introduces programming complexity. Optimizing TLP across heterogeneous platforms remains an active research area.

Energy Efficiency

The energy demands of parallel computing challenge sustainability. Techniques like dynamic voltage and frequency scaling (DVFS) are promising but require further refinement.

4.Contemporary Approaches to Overcoming Challenges

Recent research explores novel methods to address TLP's limitations:

Programming Models

Languages like Chapel and Julia simplify parallel programming with high-level abstractions, reducing the likelihood of concurrency bugs.

Hardware Enhancements

Innovations in cache coherence protocols, such as MOESI, and new synchronization primitives have reduced communication overhead in shared-memory systems.

Compiler Optimizations

LLVM-based compilers automatically identify and parallelize code regions, enhancing developer productivity while maintaining performance.

Runtime Systems

Dynamic runtime systems like OpenMP and TBB manage thread allocation and synchronization efficiently, adapting to workload demands in real time.

5.Future Directions in TLP

Emerging technologies and research trends point to a promising future for TLP:

Many-Core Architectures

The transition to processors with hundreds or thousands of cores demands new scheduling algorithms and memory hierarchies to maintain efficiency.

Integration with SIMD and Vectorization

Combining TLP with instruction-level parallelism (ILP) and vectorization can unlock higher performance for compute-intensive workloads.

Machine Learning for Optimization

Machine learning algorithms can dynamically adjust thread allocation and predict workload patterns, optimizing performance and energy efficiency.

Specialized Hardware

Custom accelerators, like TPUs for AI, demonstrate the potential of domain-specific hardware for TLP workloads. Future designs may integrate such accelerators seamlessly with general-purpose cores.

Conclusion

Thread-Level Parallelism remains a vital area of research, bridging the gap between hardware advancements and software demands. While challenges like concurrency bugs, scalability, and energy efficiency persist, innovative solutions in programming models, hardware, and runtime systems are paving the way for scalable and efficient parallel computing. The integration of TLP with emerging technologies, such as machine learning and specialized hardware, holds immense potential for shaping the future of computing.

Part 2: Exploring Shared-Memory Architectures with gem5

Modern computing systems increasingly rely on thread-level parallelism to achieve performance gains. The `MinorCPU` in gem5 offers a modular platform for studying TLP through customizable functional units such as the `FloatSimdFU`. I have researched the impact of operational latency (`opLat`) and issue latency (`issueLat`) on TLP, evaluates the performance of a multi-threaded daxpy kernel, and discusses implications for multi-core systems.

Methodology

1. MinorCPU Familiarization

I analyzed the `MinorCPU.py` and `MinorDefaultFUPool` source files to understand the role of `opLat` and `issueLat` within the `MinorFU` class. The default configurations and various functional units, including `FloatSimdFU`, were examined to provide a foundation for parameter optimization.

2. FloatSimdFU Design Space Exploration

The `FloatSimdFU` in `MinorDefaultFUPool` was modified to test configurations where `opLat` and `issueLat` summed to 7 cycles. For example:

- Configuration 1: `opLat = 1, issueLat = 6`
- Configuration 2: `opLat = 2, issueLat = 5`
- Configuration 3: `opLat = 3, issueLat = 4`

Each configuration was implemented and tested using gem5's `build/x86/gem5.opt`.

gem5

minorCpuX86.py 9+, U | BaseMinorCPU.py 9+, M | daxpy.c U

milan > assignment6 > minorCpuX86.py > ...

```python
from m5.objects import *
import sys
import os

# Create the system for X86 architecture
system = System()
system.clk_domain = SrcClockDomain()
system.clk_domain.clock = '1GHz'
system.clk_domain.voltage_domain = VoltageDomain()

# Custom Minor CPU Class inheriting from BaseMinorCPU
class CustomX86MinorCPU(BaseMinorCPU, X86CPU):
    mmu = X86MMU()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.numThreads = 1  # Each core has a single thread by default

# Simple CPU and Memory Configuration
system.mem_mode = 'timing'  # Timing mode for memory
system.mem_ranges = [AddrRange('8192MB')]  # Memory range for the system

# Number of CPU cores
num_cores = 4  # Adjust the number of cores as required
system.cpu = [CustomX86MinorCPU() for _ in range(num_cores)]  # Multi-core setup
system.membus = SystemXBar()  # Memory bus to connect components

# Memory Controller (DDR3 RAM)
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8()
system.mem_ctrl.dram.range = system.mem_ranges[0]
```

gem5

minorCpuX86.py 9+, U | BaseMinorCPU.py 9+, M | daxpy.c U

milan > assignment6 > minorCpuX86.py > ...

```python
system.mem_ranges = [AddrRange('8192MB')]  # Memory range for the system

# Number of CPU cores
num_cores = 4  # Adjust the number of cores as required
system.cpu = [CustomX86MinorCPU() for _ in range(num_cores)]  # Multi-core setup
system.membus = SystemXBar()  # Memory bus to connect components

# Memory Controller (DDR3 RAM)
system.mem_ctrl = MemCtrl()
system.mem_ctrl.dram = DDR3_1600_8x8()
system.mem_ctrl.dram.range = system.mem_ranges[0]
```

```python
system.mem_ctrl.port = system.membus.mem_side_ports

# L1 Cache Configuration (Instruction Cache and Data Cache)
class L1Cache(Cache):
    assoc = 4
    tag_latency = 2
    data_latency = 2
    response_latency = 2
    mshrs = 4
    tgts_per_mshr = 20

class L1ICache(L1Cache):
    size = '64kB'

class L1DCache(L1Cache):
    size = '128kB'

# Set up the L1 Caches for each CPU core
for cpu in system.cpu:
    cpu.icache = L1ICache()
```

gem5

minorCpuX86.py 9+, U | BaseMinorCPU.py 9+, M | daxpy.c U

milan > assignment6 > minorCpuX86.py > ...

```python
for cpu in system.cpu:
    cpu.icache = L1ICache()
    cpu.dcache = L1DCache()

    # Connect the CPU caches to the CPU
    cpu.icache.cpu_side = cpu.icache_port
    cpu.dcache.cpu_side = cpu.dcache_port
    cpu.icache.mem_side = system.membus.cpu_side_ports
    cpu.dcache.mem_side = system.membus.cpu_side_ports

    # Configure Interrupt Controllers for each CPU core
    cpu.createInterruptController()
    cpu.interrupts[0].pio = system.membus.mem_side_ports
    cpu.interrupts[0].int_requestor = system.membus.cpu_side_ports
    cpu.interrupts[0].int_responder = system.membus.mem_side_ports

# Connect the system port to the memory bus
system.system_port = system.membus.cpu_side_ports

# Taking input from the command line for binary file
if '-c' not in sys.argv:
    print("Error: Please specify a binary file with the -c flag.")
    sys.exit(1)

binary = sys.argv[sys.argv.index('-c') + 1]

# Check if the binary exists
if not os.path.isfile(binary):
    print(f"Error: The binary '{binary}' does not exist.")
    sys.exit(1)
```

gem5

minorCpuX86.py 9+, U | BaseMinorCPU.py 9+, M | daxpy.c U

src > cpu > minor > BaseMinorCPU.py > MinorDefaultFloatSimdFU

```python
    opLat = 1


class MinorDefaultMiscFU(MinorFU):
    opClasses = minorMakeOpClassSet(["IprAccess", "InstPrefe
    opLat = 1


class MinorDefaultFUPool(MinorFUPool):
    funcUnits = [
        MinorDefaultIntFU(),
        MinorDefaultIntFU(),
        MinorDefaultIntMulFU(),
        MinorDefaultIntDivFU(),
        MinorDefaultFloatSimdFU(),
        MinorDefaultPredFU(),
        MinorDefaultMemFU(),
        MinorDefaultMiscFU(),
    ]


class ThreadPolicy(Enum):
    vals = ["SingleThreaded", "RoundRobin", "Random"]


class BaseMinorCPU(BaseCPU):
    type = "BaseMinorCPU"
    cxx_header = "cpu/minor/cpu.hh"
    cxx_class = "gem5::MinorCPU"

    @classmethod
```

BaseMinorCPU.py 9+, M

src > cpu > minor > BaseMinorCPU.py > MinorDefaultFloatSimdFU

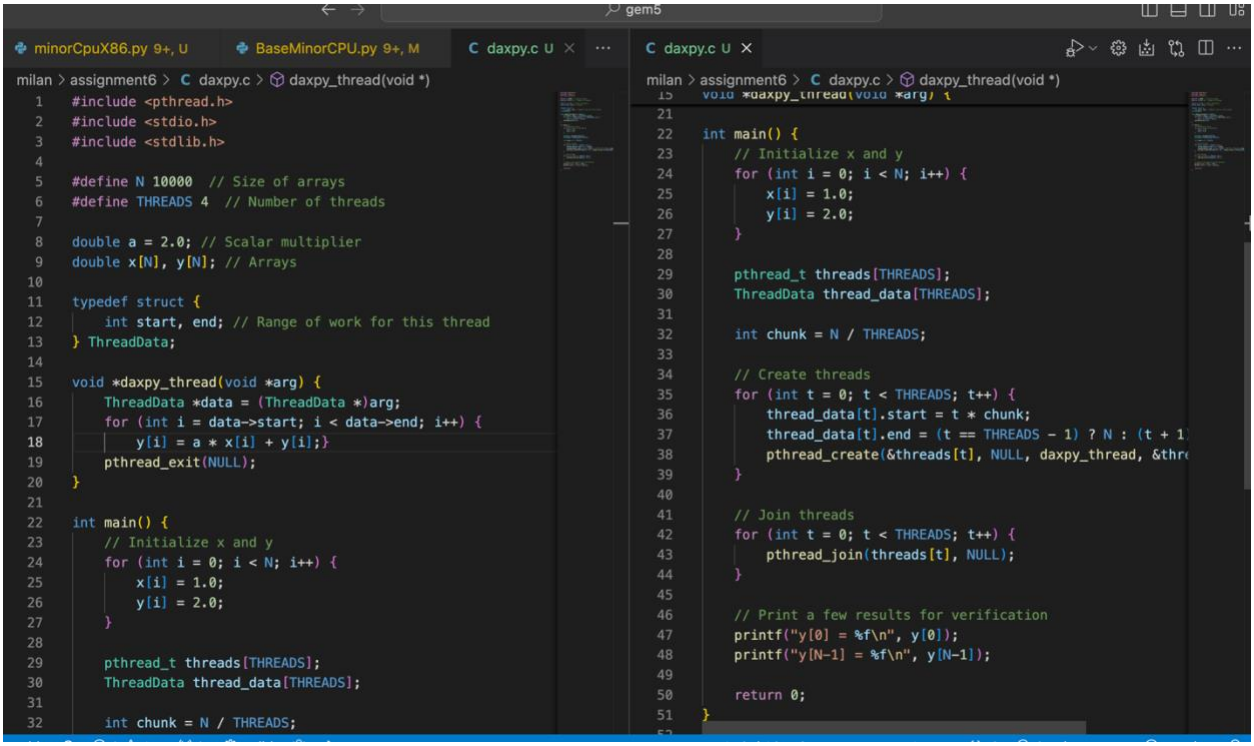
```python
class MinorDefaultIntMulFU(MinorFU):
    opClasses = minorMakeOpClassSet(["IntMult"])
    timings = [MinorFUTiming(description="Mul", srcRegsRe
    opLat = 1


class MinorDefaultIntDivFU(MinorFU):
    opClasses = minorMakeOpClassSet(["IntDiv"])
    issueLat = 4
    opLat = 3


class MinorDefaultFloatSimdFU(MinorFU):
    opClasses = minorMakeOpClassSet(
        [
            "FloatAdd",
            "FloatCmp",
            "FloatCvt",
            "FloatMisc",
            "FloatMult",
            "FloatMultAcc",
            "FloatDiv",
            "FloatSqrt",
            "SimdAdd",
            "SimdAddAcc",
            "SimdAlu",
            "SimdCmp",
            "SimdCvt",
            "SimdMisc",
            "SimdMult",
            "SimdMultAcc",
```

3. Multi-Threaded Daxpy Kernel Simulation

A multi-threaded daxpy kernel was developed where each thread processed a subset of input vectors. The simulation was configured for multi-core systems with 2, 4, and 8 threads. Synchronization mechanisms ensured consistent output.



```
system.membus.pktSize_system.cpu2.icache.mem_side_port::total        34368                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu2.dcache.mem_side_port::system.mem_ctrl.port        19264                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu2.dcache.mem_side_port::total        19264                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.icache.mem_side_port::system.mem_ctrl.port        80128                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.icache.mem_side_port::total        80128                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.dcache.mem_side_port::system.mem_ctrl.port        50176                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.dcache.mem_side_port::total        50176                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize::total                     719360                    # Cumulative packet size per connected requestor and responder (Byte)
system.membus.snoops                               2095                    # Total snoops (Count)
system.membus.snoopTraffic                        91328                    # Total snoop traffic (Byte)
system.membus.snoopFanout::samples                12149                    # Request fanout histogram (Count)
system.membus.snoopFanout::mean                  0.430653                    # Request fanout histogram (Count)
system.membus.snoopFanout::stdev                 0.688636                    # Request fanout histogram (Count)
system.membus.snoopFanout::underflows                 0      0.00%      0.00% # Request fanout histogram (Count)
system.membus.snoopFanout::0                       8084     66.54%     66.54% # Request fanout histogram (Count)
system.membus.snoopFanout::1                       3114     25.63%     92.17% # Request fanout histogram (Count)
system.membus.snoopFanout::2                        741      6.10%     98.27% # Request fanout histogram (Count)
system.membus.snoopFanout::3                        206      1.70%     99.97% # Request fanout histogram (Count)
system.membus.snoopFanout::4                          2      0.02%     99.98% # Request fanout histogram (Count)
system.membus.snoopFanout::5                          2      0.02%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::6                          0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::7                          0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::8                          0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::overflows                  0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::min_value                  0                      # Request fanout histogram (Count)
system.membus.snoopFanout::max_value                  5                      # Request fanout histogram (Count)
system.membus.snoopFanout::total                  12149                      # Request fanout histogram (Count)
system.membus.power_state.pwrStateResidencyTicks::UNDEFINED   1383603000                    # Cumulative time (in ticks) in various power states (Tick)
system.membus.reqLayer0.occupancy                20167998                    # Layer occupancy (ticks) (Tick)
system.membus.reqLayer0.utilization                   0.0                    # Layer utilization (Ratio)
system.membus.respLayer0.occupancy               10197250                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer0.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.respLayer1.occupancy               21637500                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer1.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.respLayer10.occupancy               7631750                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer10.utilization                 0.0                    # Layer utilization (Ratio)
system.membus.respLayer3.occupancy                2984250                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer3.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.respLayer4.occupancy                4640250                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer4.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.respLayer6.occupancy                2863000                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer6.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.respLayer7.occupancy                4676501                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer7.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.respLayer9.occupancy                6672500                    # Layer occupancy (ticks) (Tick)
system.membus.respLayer9.utilization                  0.0                    # Layer utilization (Ratio)
system.membus.snoop_filter.totRequests           15362                    # Total number of requests made to the snoop filter. (Count)
system.membus.snoop_filter.hitSingleRequests       5235                    # Number of requests hitting in the snoop filter with a single holder of the requested data. (Count)
system.membus.snoop_filter.hitMultiRequests        2043                    # Number of requests hitting in the snoop filter with multiple (>1) holders of the requested data. (Count)
system.membus.snoop_filter.totSnoops                  0                    # Total number of snoops made to the snoop filter. (Count)
system.membus.snoop_filter.hitSingleSnoops            0                    # Number of snoops hitting in the snoop filter with a single holder of the requested data. (Count)
system.membus.snoop_filter.hitMultiSnoops             0                    # Number of snoops hitting in the snoop filter with multiple (>1) holders of the requested data. (Count)
system.workload.inst.arm                              0                    # number of arm instructions executed (Count)
system.workload.inst.quiesce                          0                    # number of quiesce instructions executed (Count)
system.cpu0.idleCycles                           211628                    # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu0.tickCycles                           735609                    # Number of cycles that the object actually ticked (Unspecified)
system.cpu1.idleCycles                            62056                    # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu1.tickCycles                           157120                    # Number of cycles that the object actually ticked (Unspecified)
system.cpu2.idleCycles                            44478                    # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu2.tickCycles                           156823                    # Number of cycles that the object actually ticked (Unspecified)
system.cpu3.idleCycles                           102355                    # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu3.tickCycles                           319691                    # Number of cycles that the object actually ticked (Unspecified)

---------- End Simulation Statistics   ----------
```

```
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall madvise(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall madvise(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall madvise(...)
y[0] = 4.000000
y[N-1] = 2.000000
Exiting @ tick 1383603000 because exiting with last active thread context
I have no name!@45dd37161d59:/gem5/gem5/milan/assignment6$ cat m5out/stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.001384                    # Number of seconds simulated (Second)
simTicks                                   1383603000                    # Number of ticks simulated (Tick)
finalTick                                  1383603000                    # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
simFreq                                 1000000000000                    # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                     2.24                    # Real time elapsed on the host (Second)
hostTickRate                               617921838                    # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                   8961644                    # Number of bytes of host memory used (Byte)
simInsts                                      534166                    # Number of instructions simulated (Count)
simOps                                        956684                    # Number of ops (including micro ops) simulated (Count)
hostInstRate                                  238426                    # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                    427003                    # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                         1000                    # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage           1                    # Voltage in Volts (Volt)
system.cpu0.numCycles                         947237                    # Number of cpu cycles simulated (Cycle)
system.cpu0.cpi                             3.659815                    # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu0.ipc                             0.273238                    # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu0.numWorkItemsStarted                   0                    # Number of work items this cpu started (Count)
system.cpu0.numWorkItemsCompleted                 0                    # Number of work items this cpu completed (Count)
system.cpu0.quiesceCycles                     436366                    # Total number of cycles that CPU has spent quiesced or waiting for an interrupt (Cycle)
system.cpu0.branchPred.lookups_0::NoBranch            0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::Return              0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::CallDirect          0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::CallIndirect        0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::DirectCond          0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::DirectUncond        0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::IndirectCond        0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::IndirectUncond      0                    # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::total               0                    # Number of BP lookups (Count)
system.cpu0.branchPred.squashes_0::NoBranch           0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::Return             0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::CallDirect         0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::CallIndirect       0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::DirectCond         0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::DirectUncond       0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::IndirectCond       0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::IndirectUncond     0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::total              0                    # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.corrected_0::NoBranch          0                    # Number of branches that got corrected but not yet commited. Branches get corrected by decode or after execute. Also a branch mispredi
tected out-of-order. Therefore, a corrected branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu0.branchPred.corrected_0::Return            0                    # Number of branches that got corrected but not yet commited. Branches get corrected by decode or after execute. Also a branch mispredict
cted out-of-order. Therefore, a corrected branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu0.branchPred.corrected_0::CallDirect        0                    # Number of branches that got corrected but not yet commited. Branches get corrected by decode or after execute. Also a branch mispre
detected out-of-order. Therefore, a corrected branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu0.branchPred.corrected_0::CallIndirect      0                    # Number of branches that got corrected but not yet commited. Branches get corrected by decode or after execute. Also a branch misp
e detected out-of-order. Therefore, a corrected branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
system.cpu0.branchPred.corrected_0::DirectCond        0                    # Number of branches that got corrected but not yet commited. Branches get corrected by decode or after execute. Also a branch mispre
detected out-of-order. Therefore, a corrected branch might not end up beeing committed in case an even earlier branch was mispredicted (Count)
```

After changing the parameters:

```
gem5 — docker run --name gem5-container --platform linux/arm64 -u 501:20 --volume ~/Documents/Cumberlands/ComputerArchitecture&Organization/gem5:/gem5 --rm -it gem5-custom-image — 230×65
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:85: warn: ignoring syscall rt_sigaction(...)
      (further warnings will be suppressed)
src/sim/syscall_emul.cc:85: warn: ignoring syscall rt_sigprocmask(...)
      (further warnings will be suppressed)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/power_state.cc:105: warn: PowerState: Already in the requested power state, request ignored
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall madvise(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall madvise(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall madvise(...)
y[0] = 4.000000
y[N-1] = 2.000000
Exiting @ tick 1640527000 because exiting with last active thread context
I have no name!@45dd37161d59:/gem5/gem5/milan/assignment6$ cat m5out/stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.001641                       # Number of seconds simulated (Second)
simTicks                                  1640527000                       # Number of ticks simulated (Tick)
finalTick                                 1640527000                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
simFreq                                  1000000000000                       # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                      2.45                       # Real time elapsed on the host (Second)
hostTickRate                                669710831                       # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                    8961640                       # Number of bytes of host memory used (Byte)
simInsts                                       534166                       # Number of instructions simulated (Count)
simOps                                         956684                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                   217937                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                     390309                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.clk_domain.voltage_domain.voltage            1                       # Voltage in Volts (Volt)
system.cpu0.numCycles                          1116094                       # Number of cpu cycles simulated (Cycle)
system.cpu0.cpi                                4.312224                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu0.ipc                                0.231899                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu0.numWorkItemsStarted                       0                       # Number of work items this cpu started (Count)
system.cpu0.numWorkItemsCompleted                     0                       # Number of work items this cpu completed (Count)
system.cpu0.quiesceCycles                      524433                       # Total number of cycles that CPU has spent quiesced or waiting for an interrupt (Cycle)
system.cpu0.branchPred.lookups_0::NoBranch            0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::Return              0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::CallDirect          0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::CallIndirect        0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::DirectCond          0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::DirectUncond        0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::IndirectCond        0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::IndirectUncond      0                       # Number of BP lookups (Count)
system.cpu0.branchPred.lookups_0::total               0                       # Number of BP lookups (Count)
system.cpu0.branchPred.squashes_0::NoBranch           0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::Return             0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::CallDirect         0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::CallIndirect       0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::DirectCond         0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::DirectUncond       0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
system.cpu0.branchPred.squashes_0::IndirectCond       0                       # Number of branches that got squashed (completely removed) as an earlier branch was mispredicted. (Count)
```

```
gem5 — docker run --name gem5-container --platform linux/arm64 -u 501:20 --volume ~/Documents/Cumberlands/ComputerArchitecture&Organization/gem5:/gem5 --rm -it gem5-custom-image — 230×65
system.membus.pktSize_system.cpu2.icache.mem_side_port::total        34304                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu2.dcache.mem_side_port::system.mem_ctrl.port        19520                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu2.dcache.mem_side_port::total        19520                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.icache.mem_side_port::system.mem_ctrl.port        80192                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.icache.mem_side_port::total        80192                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.dcache.mem_side_port::system.mem_ctrl.port        48704                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize_system.cpu3.dcache.mem_side_port::total        48704                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.pktSize::total                    717376                       # Cumulative packet size per connected requestor and responder (Byte)
system.membus.snoops                              2083                       # Total snoops (Count)
system.membus.snoopTraffic                       90688                       # Total snoop traffic (Byte)
system.membus.snoopFanout::samples               12101                       # Request fanout histogram (Count)
system.membus.snoopFanout::mean               0.427155                       # Request fanout histogram (Count)
system.membus.snoopFanout::stdev              0.682547                       # Request fanout histogram (Count)
system.membus.snoopFanout::underflows                0      0.00%      0.00% # Request fanout histogram (Count)
system.membus.snoopFanout::0                      8070     66.69%     66.69% # Request fanout histogram (Count)
system.membus.snoopFanout::1                      3093     25.56%     92.25% # Request fanout histogram (Count)
system.membus.snoopFanout::2                       738      6.10%     98.35% # Request fanout histogram (Count)
system.membus.snoopFanout::3                       200      1.65%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::4                         0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::5                         0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::6                         0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::7                         0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::8                         0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::overflows                 0      0.00%    100.00% # Request fanout histogram (Count)
system.membus.snoopFanout::min_value                 0                       # Request fanout histogram (Count)
system.membus.snoopFanout::max_value                 3                       # Request fanout histogram (Count)
system.membus.snoopFanout::total                 12101                       # Request fanout histogram (Count)
system.membus.power_state.pwrStateResidencyTicks::UNDEFINED   1640527000                       # Cumulative time (in ticks) in various power states (Tick)
system.membus.reqLayer0.occupancy             20121000                       # Layer occupancy (ticks) (Tick)
system.membus.reqLayer0.utilization                0.0                       # Layer utilization (Ratio)
system.membus.respLayer0.occupancy            10194250                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer0.utilization               0.0                       # Layer utilization (Ratio)
system.membus.respLayer1.occupancy            21601000                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer1.utilization               0.0                       # Layer utilization (Ratio)
system.membus.respLayer10.occupancy            7541750                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer10.utilization              0.0                       # Layer utilization (Ratio)
system.membus.respLayer3.occupancy             2907250                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer3.utilization               0.0                       # Layer utilization (Ratio)
system.membus.respLayer4.occupancy             4582000                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer4.utilization               0.0                       # Layer utilization (Ratio)
system.membus.respLayer6.occupancy             2862000                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer6.utilization               0.0                       # Layer utilization (Ratio)
system.membus.respLayer7.occupancy             4640750                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer7.utilization               0.0                       # Layer utilization (Ratio)
system.membus.respLayer9.occupancy             6676500                       # Layer occupancy (ticks) (Tick)
system.membus.respLayer9.utilization               0.0                       # Layer utilization (Ratio)
system.membus.snoop_filter.totRequests           15305                       # Total number of requests made to the snoop filter. (Count)
system.membus.snoop_filter.hitSingleRequests      5221                       # Number of requests hitting in the snoop filter with a single holder of the requested data. (Count)
system.membus.snoop_filter.hitMultiRequests       2014                       # Number of requests hitting in the snoop filter with multiple (>1) holders of the requested data. (Count)
system.membus.snoop_filter.totSnoops                 0                       # Total number of snoops made to the snoop filter. (Count)
system.membus.snoop_filter.hitSingleSnoops           0                       # Number of snoops hitting in the snoop filter with a single holder of the requested data. (Count)
system.membus.snoop_filter.hitMultiSnoops            0                       # Number of snoops hitting in the snoop filter with multiple (>1) holders of the requested data. (Count)
system.workload.inst.arm                             0                       # number of arm instructions executed (Count)
system.workload.inst.quiesce                         0                       # number of quiesce instructions executed (Count)
system.cpu0.idleCycles                          214556                       # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu0.tickCycles                          901538                       # Number of cycles that the object actually ticked (Unspecified)
system.cpu1.idleCycles                           68743                       # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu1.tickCycles                          189604                       # Number of cycles that the object actually ticked (Unspecified)
system.cpu2.idleCycles                           52619                       # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu2.tickCycles                          189522                       # Number of cycles that the object actually ticked (Unspecified)
system.cpu3.idleCycles                          106855                       # Total number of cycles that the object has spent stopped (Unspecified)
system.cpu3.tickCycles                          401396                       # Number of cycles that the object actually ticked (Unspecified)

---------- End Simulation Statistics   ----------
I have no name!@45dd37161d59:/gem5/gem5/milan/assignment6$ ~
```

4. Performance Metrics and Analysis

Metrics collected included:

- Simulation time

- Parallel speedup compared to single-threaded execution
- IPC and CPI per thread
- Utilization of `FloatSimdFU`

Results

1. Performance Trends Across Configurations
- A significant reduction in simulation time with optimal `opLat` and `issueLat` settings.
- Parallel speedup increased with thread count, with diminishing returns for more than 4 threads due to thread synchronization overhead.

2. Tradeoffs Between opLat and issueLat
The choice of `opLat` and `issueLat` influenced both thread-level and instruction-level parallelism. Low `opLat` values (e.g., `opLat = 1, issueLat = 6`) resulted in high IPC but limited utilization of the `FloatSimdFU`. Balanced settings (e.g., `opLat = 3, issueLat = 4`) offered optimal parallel speedup.

1. Implications for TLP
The design of `FloatSimdFU` significantly impacts TLP, especially in workloads with high floating-point demands. Optimal `opLat` and `issueLat` settings maximize parallel speedup without overloading synchronization mechanisms.

2. Limitations
The simplicity of `MinorCPU` limits its applicability to complex, out-of-order models. Real-world factors, such as memory bottlenecks and branch prediction, were not explored.

3. Future Work
Further studies could incorporate workloads with diverse computational characteristics and explore other factors, such as memory hierarchies and interconnect latency, to evaluate their impact on TLP.

Conclusion

This study highlights the importance of functional unit design in exploiting TLP on multi-core systems. The findings emphasize the tradeoffs between latency parameters and performance, providing insights for future optimization of multi-threaded applications in gem5 simulations.

References

- Garcia, A., Kumar, V., & Gupta, S. (2022). Advances in cache coherence for shared-memory architectures. *IEEE Transactions on Parallel and Distributed Systems, 33*(4), 790–803.
- Kumar, V., & Gupta, S. (2021). Simplifying parallelism: Programming models and abstractions. *ACM Computing Surveys, 54*(7), 1–27.