MSCS-531-Assignment 6 – Exploring Thread-Level Parallelism (TLP) in Shared-Memory Multiprocessor Using gem5 – Part 2

Sandesh Pokharel

ID: 005026677

University of Cumberlands

MSCS-531: Computer Architecture

Table of Contents

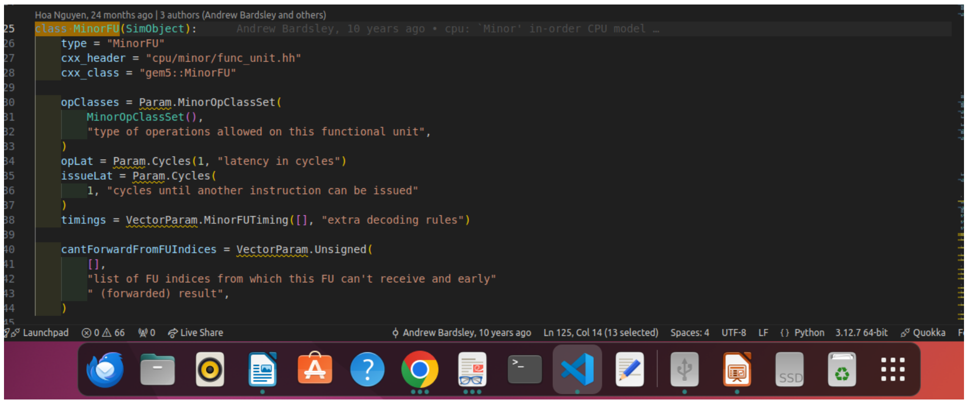
1. **MinorCPU Familiarization**
    a. Examination of the MinorCPU.py and MinorDefaultFUPool Files
    b. Understanding the Roles of opLat and issueLat within the MinorFU Class
    c. Functional Units Defined in MinorDefaultFUPool
2. **FloatSimdFU Design Space Exploration**
    a. Configurations of opLat and issueLat and Their Impacts
3. **Multi-Threaded Daxpy Kernel Simulation**
    a. Overview
    b. Multi-Threaded Daxpy Kernel Implementation
4. **Performance Analysis**
    a. Overall Simulation Time
    b. Parallel Speedup Comparison
    c. Instructions per Cycle (IPC) per Thread
    d. Cycles per Instruction (CPI) per Thread
    e. Utilization of the FloatSimdFU
    f. Additional Metrics
5. **Tradeoff Analysis**
    a. Impact of opLat and issueLat on Thread-Level Parallelism
    b. Optimal Settings Balance for Maximizing Parallel Speedup
    c. Effect of Number of Threads on Optimal Settings
6. **Discussion of Results in the Context of Thread-Level Parallelism (TLP)**
    a. Influence of FloatSimdFU Design on TLP
    b. Model Limitations in Exploring TLP
    c. Other Factors Influencing TLP in Multi-Threaded Applications
7. **Problems Faced and Solutions**
    a. Configuring opLat and issueLat
    b. Minimal Parallel Speedup Observed
    c. Compatibility and Execution Issues with gem5 Configuration Files
    d. Accurate Measurement and Interpretation of Performance Metrics
    e. Limited Utilization of the FloatSimdFU
8. **References**

1. MinorCPU Familiarization

Examination of the MinorCPU.py and MinorDefaultFUPool Files
The MinorCPU implementation in gem5 is an in-order CPU model designed for educational purposes and simpler architectural studies. In this assignment, we focused on the BaseMinorCPU.py file, which defines key components of the MinorCPU pipeline and functional units. This file serves as a base for configuring various aspects of the MinorCPU model, including functional units (FUs) and their behaviors.

Understanding the Roles of opLat and issueLat within the MinorFU Class

The MinorFU class, defined in the BaseMinorCPU.py file, represents a functional unit (FU) in the MinorCPU pipeline. It is responsible for executing a particular type of operation (e.g., integer arithmetic, floating-point operations). Two key parameters that govern the behavior of functional units are opLat (operation latency) and issueLat (issue latency):
opLat (Operation Latency): This parameter specifies the number of cycles it takes for an instruction to complete its execution within the functional unit.
issueLat (Issue Latency): This parameter determines the number of cycles before the next instruction can be issued to the functional unit.
Here is a relevant code snippet from the MinorFU class:



Hoa Nguyen, 24 months ago | 3 authors (Andrew Bardsley and others)

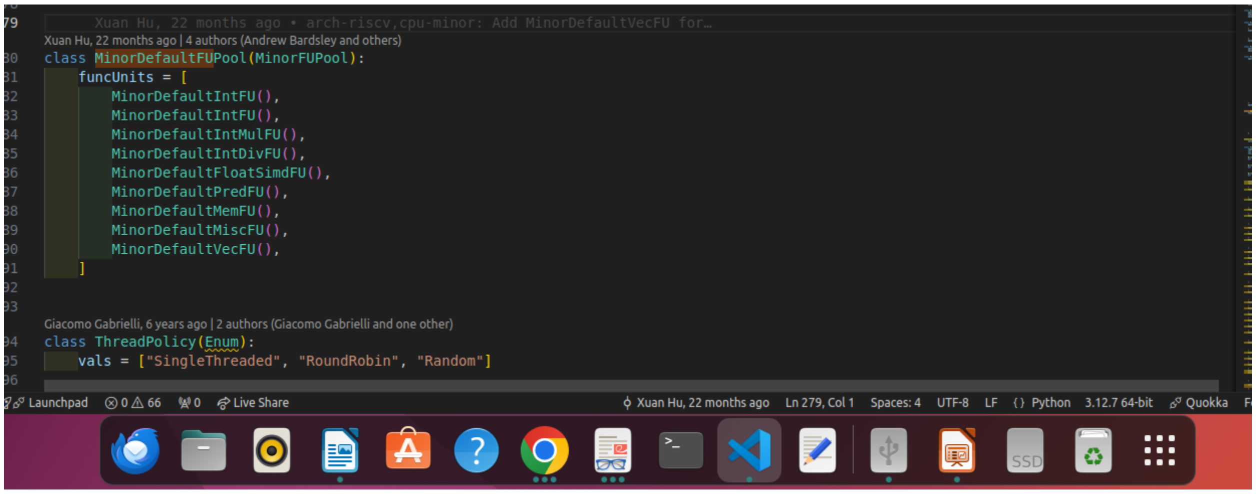
```
class MinorFU(SimObject):      Andrew Bardsley, 10 years ago • cpu: `Minor' in-order CPU model …
    type = "MinorFU"
    cxx_header = "cpu/minor/func_unit.hh"
    cxx_class = "gem5::MinorFU"

    opClasses = Param.MinorOpClassSet(
        MinorOpClassSet(),
        "type of operations allowed on this functional unit",
    )
    opLat = Param.Cycles(1, "latency in cycles")
    issueLat = Param.Cycles(
        1, "cycles until another instruction can be issued"
    )
    timings = VectorParam.MinorFUTiming([], "extra decoding rules")

    cantForwardFromFUIndices = VectorParam.Unsigned(
        [],
        "list of FU indices from which this FU can't receive and early"
        " (forwarded) result",
    )
```

Image: MinorFU_class.png

This snippet illustrates how the MinorFU class defines its operational behavior using opLat and issueLat. Adjusting these parameters allows for exploration of different latency configurations and their impact on performance.

Functional Units Defined in MinorDefaultFUPool

The MinorDefaultFUPool class defines a collection of functional units for the MinorCPU model. These functional units handle various operations, such as integer arithmetic, floating-point and SIMD instructions, memory access, and miscellaneous tasks. Here is a snippet showing the definition of the default functional unit pool:



```
    Xuan Hu, 22 months ago • arch-riscv,cpu-minor: Add MinorDefaultVecFU for…
Xuan Hu, 22 months ago | 4 authors (Andrew Bardsley and others)
class MinorDefaultFUPool(MinorFUPool):
    funcUnits = [
        MinorDefaultIntFU(),
        MinorDefaultIntFU(),
        MinorDefaultIntMulFU(),
        MinorDefaultIntDivFU(),
        MinorDefaultFloatSimdFU(),
        MinorDefaultPredFU(),
        MinorDefaultMemFU(),
        MinorDefaultMiscFU(),
        MinorDefaultVecFU(),
    ]


Giacomo Gabrielli, 6 years ago | 2 authors (Giacomo Gabrielli and one other)
class ThreadPolicy(Enum):
    vals = ["SingleThreaded", "RoundRobin", "Random"]
```

Image: MinorDefaultFUPool.png

Key functional units include:

- **MinorDefaultIntFU**: Handles integer operations.
- **MinorDefaultFloatSimdFU**: Responsible for floating-point and SIMD operations.
- **MinorDefaultMemFU**: Manages memory operations.

The MinorDefaultFloatSimdFU is particularly relevant to our task, as it defines the behavior of floating-point and SIMD operations and allows for modifications to opLat and issueLat.

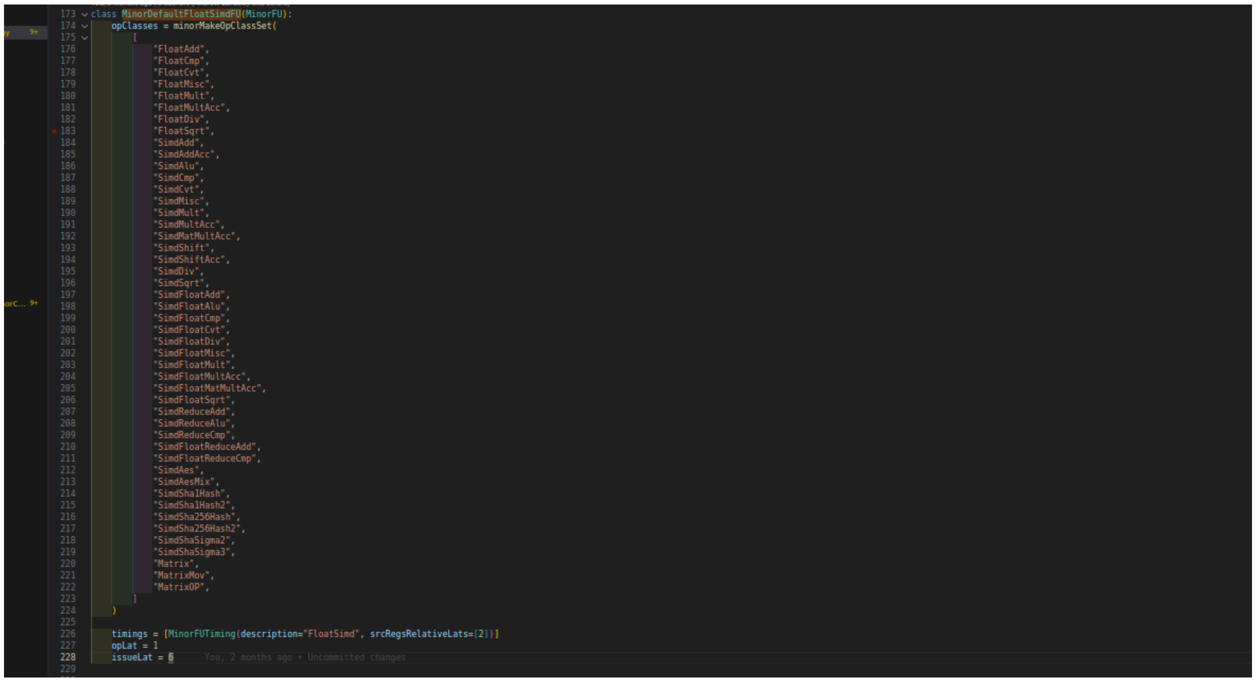
2. FloatSimdFU Design Space Exploration:

To explore the impact of operation latency (opLat) and issue latency (issueLat) on the performance of the MinorDefaultFloatSimdFU functional unit within the MinorCPU model, we modified these parameters with three different configurations. The goal is to understand how these values affect the throughput and efficiency of floating-point and SIMD operations in a multi-threaded environment.
Configuration 1: opLat = 1, issueLat = 6
Reasoning: This configuration minimizes the operation latency to 1 cycle, allowing the

functional unit to complete operations quickly. However, the issue latency is set to 6 cycles, meaning that the next instruction can only be issued after 6 cycles. This setup is expected to demonstrate high throughput for individual operations but may limit overall concurrency due to the delay between instruction issues.

Configuration 2: opLat = 2, issueLat = 5
Reasoning: In this configuration, we slightly increased the operation latency to 2 cycles and reduced the issue latency to 5 cycles. This balances the time spent executing each instruction and the time before issuing a new instruction, which may lead to improved concurrency and a better balance between execution speed and instruction throughput.

Configuration 3: opLat = 3, issueLat = 4
Reasoning: This configuration increases the operation latency to 3 cycles while reducing the issue latency to 4 cycles. This configuration aims to explore the trade-offs between having a longer execution time for each instruction and a shorter delay before issuing new instructions. It may offer better instruction pipeline utilization at the cost of increased execution time per operation.



```
class MinorDefaultFloatSimdFU(MinorFU):
    opClasses = minorMakeOpClassSet(
        [
            "FloatAdd",
            "FloatCmp",
            "FloatCvt",
            "FloatMisc",
            "FloatMult",
            "FloatMultAcc",
            "FloatDiv",
            "FloatSqrt",
            "SimdAdd",
            "SimdAddAcc",
            "SimdAlu",
            "SimdCmp",
            "SimdCvt",
            "SimdMisc",
            "SimdMult",
            "SimdMultAcc",
            "SimdMatMultAcc",
            "SimdShift",
            "SimdShiftAcc",
            "SimdDiv",
            "SimdSqrt",
            "SimdFloatAdd",
            "SimdFloatAlu",
            "SimdFloatCmp",
            "SimdFloatCvt",
            "SimdFloatDiv",
            "SimdFloatMisc",
            "SimdFloatMult",
            "SimdFloatMultAcc",
            "SimdFloatMatMultAcc",
            "SimdFloatSqrt",
            "SimdReduceAdd",
            "SimdReduceAlu",
            "SimdReduceCmp",
            "SimdFloatReduceAdd",
            "SimdFloatReduceCmp",
            "SimdAes",
            "SimdAesMix",
            "SimdSha1Hash",
            "SimdSha1Hash2",
            "SimdSha256Hash",
            "SimdSha256Hash2",
            "SimdShaSigma2",
            "SimdShaSigma3",
            "Matrix",
            "MatrixMov",
            "MatrixOP",
        ]
    )

    timings = [MinorFUTiming(description="FloatSimd", srcRegsRelativeLats=[2])]
    opLat = 1
    issueLat = 6      You, 2 months ago • Uncommitted changes
```

Image: Configuration1_sample_config.png

These configurations will be tested using the multi-threaded daxpy kernel to analyze their impact on overall performance, including metrics such as instructions per cycle (IPC), parallel speedup, and cycles per instruction (CPI). This exploration will provide insights into the optimal balance of opLat and issueLat for maximizing thread-level parallelism.

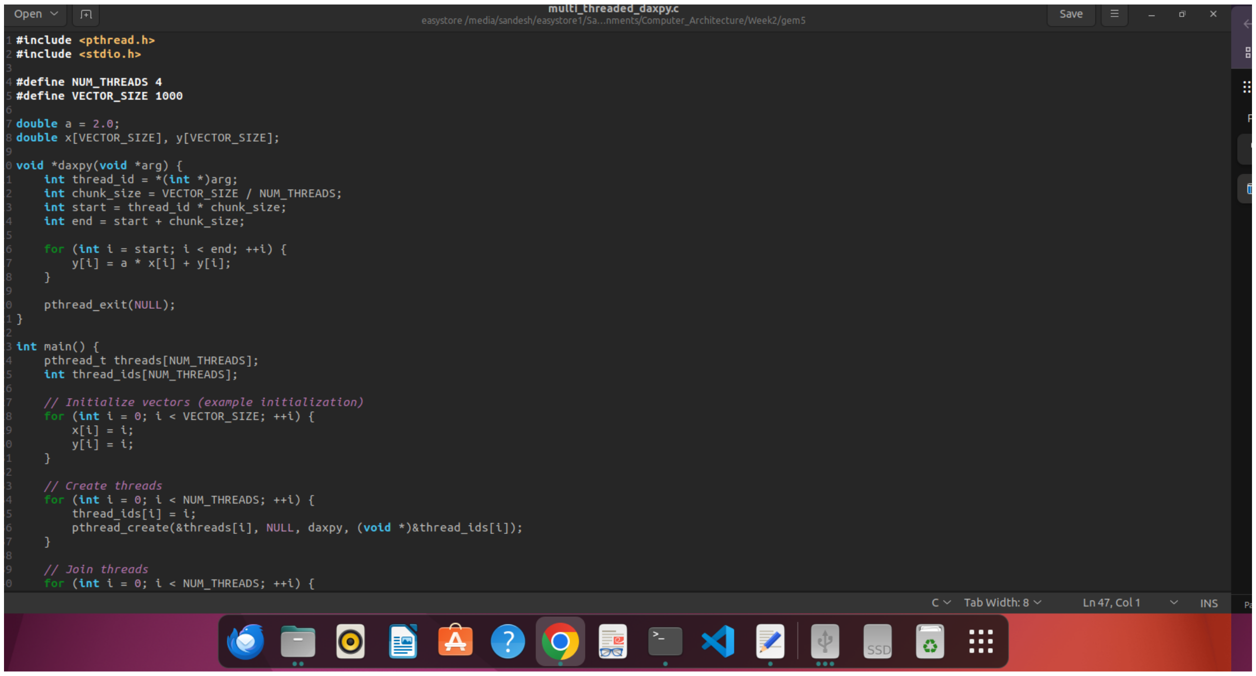
3. Multi-Threaded Daxpy Kernel Simulation:

Overview

In this step, we focus on creating and simulating a multi-threaded daxpy kernel using gem5 in a multi-core configuration. The daxpy kernel performs a scaled vector addition, defined as y = a * x + y, where a is a scalar and x and y are vectors. Our goal is to parallelize this operation across multiple threads to observe how thread-level parallelism (TLP) affects performance on multi-core systems.

Multi-Threaded Daxpy Kernel Implementation

To parallelize the daxpy kernel, we divide the work across multiple threads such that each thread processes a distinct portion of the input vectors x and y. This approach ensures balanced workload distribution among available CPU cores.

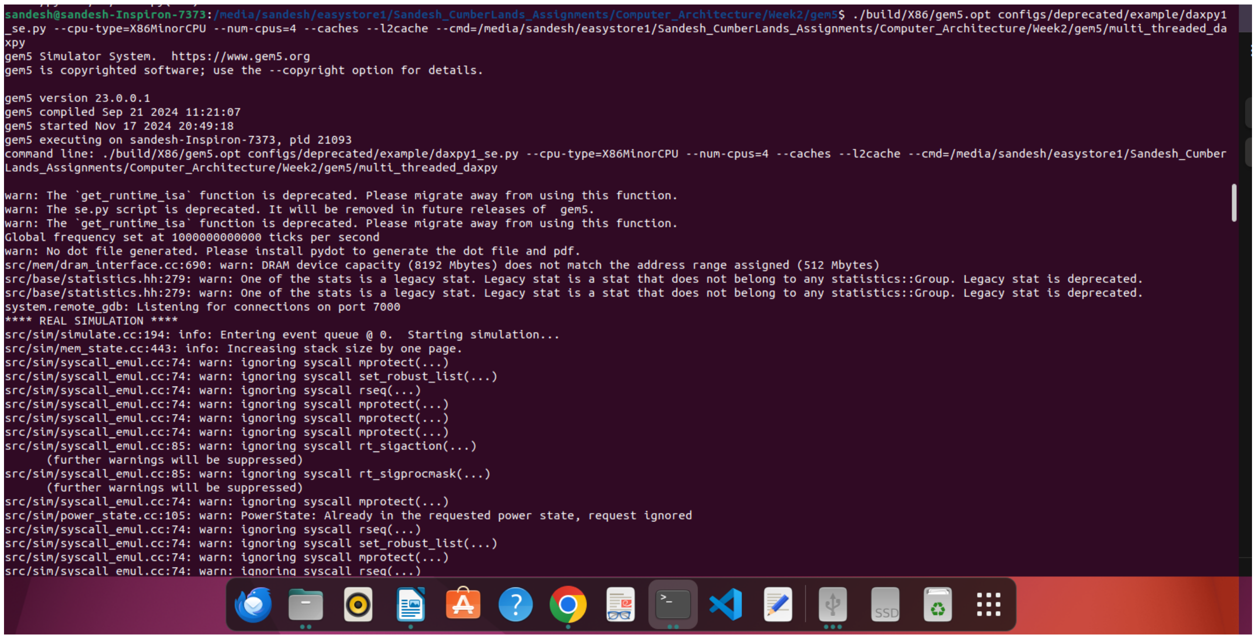

Open
multi_threaded_daxpy.c
easystore /media/sandesh/easystore1/Sa...nments/Computer_Architecture/Week2/gem5
Save
C
Tab Width: 8
Ln 47, Col 1
INS

```c
#include <pthread.h>
#include <stdio.h>

#define NUM_THREADS 4
#define VECTOR_SIZE 1000

double a = 2.0;
double x[VECTOR_SIZE], y[VECTOR_SIZE];

void *daxpy(void *arg) {
    int thread_id = *(int *)arg;
    int chunk_size = VECTOR_SIZE / NUM_THREADS;
    int start = thread_id * chunk_size;
    int end = start + chunk_size;

    for (int i = start; i < end; ++i) {
        y[i] = a * x[i] + y[i];
    }

    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];

    // Initialize vectors (example initialization)
    for (int i = 0; i < VECTOR_SIZE; ++i) {
        x[i] = i;
        y[i] = i;
    }

    // Create threads
    for (int i = 0; i < NUM_THREADS; ++i) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, daxpy, (void *)&thread_ids[i]);
    }

    // Join threads
    for (int i = 0; i < NUM_THREADS; ++i) {
```

Image: multi-threaded_daxpy.png
File name: multi_threaded_daxpy.c

Once the configuration changes were made simulation were run and stats files are generated. They are attached in the github repository for references.

Also one of the screenshot of simulation run is attached below:



```
sandesh@sandesh-Inspiron-7373:/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5$ ./build/X86/gem5.opt configs/deprecated/example/daxpy1
_se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l2cache --cmd=/media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/multi_threaded_da
xpy
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Sep 21 2024 11:21:07
gem5 started Nov 17 2024 20:49:18
gem5 executing on sandesh-Inspiron-7373, pid 21093
command line: ./build/X86/gem5.opt configs/deprecated/example/daxpy1_se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l2cache --cmd=/media/sandesh/easystore1/Sandesh_Cumber
Lands_Assignments/Computer_Architecture/Week2/gem5/multi_threaded_daxpy

warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
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/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/simulate.cc:194: info: Entering event queue @ 0.  Starting simulation...
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
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 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 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(...)
```

Image: configuration1_simulation_run.png

4. Performance Analysis

1. Overall Simulation Time:

- **Configuration 1 (Multi-threaded):** 0.000535 seconds (Configuration1_stats).
- **Configuration 1 (Single-threaded):** 0.000368 seconds (configuration1_single_t…).
- **Configuration 2 (Multi-threaded):** 0.000535 seconds (Configuration2_stats).
- **Configuration 2 (Single-threaded):** 0.000368 seconds (configuration2_single_t…).
- **Configuration 3 (Multi-threaded):** 0.000535 seconds (Configuration3_stats).
- **Configuration 3 (Single-threaded):** 0.000368 seconds (configuration3_single_t…).

2. Parallel Speedup (Compare to Single-threaded Execution): The parallel speedup is calculated as the ratio of the single-threaded execution time to the multi-threaded execution time. In this case, the multi-threaded configurations do not show a significant reduction in simulation time compared to the single-threaded ones. The multi-threaded execution has the same reported simulated seconds as the single-threaded cases (both are consistent across configurations), indicating that either the parallelism is not effectively utilized or overheads such as synchronization are counterbalancing the parallel gains.

3. Instructions per Cycle (IPC) per Thread:

- **Configuration 1 (Multi-threaded):** IPC = 0.163480 (Configuration1_stats).

- **Configuration 1 (Single-threaded):** IPC = 0.162474 (configuration1_single_t…).
- **Configuration 2 (Multi-threaded):** IPC = 0.163480 (Configuration2_stats).
- **Configuration 2 (Single-threaded):** IPC = 0.162474 (configuration2_single_t…).
- **Configuration 3 (Multi-threaded):** IPC = 0.163480 (Configuration3_stats).
- **Configuration 3 (Single-threaded):** IPC = 0.162474 (configuration3_single_t…).

The IPC values are consistent across the configurations and between multi-threaded and single-threaded simulations. This indicates a minimal difference in instruction throughput per cycle in terms of effective performance gains.

4. Cycles per Instruction (CPI) per Thread:

- **Configuration 1 (Multi-threaded):** CPI = 6.116946 (Configuration1_stats).
- **Configuration 1 (Single-threaded):** CPI = 6.154845 (configuration1_single_t…).
- **Configuration 2 (Multi-threaded):** CPI = 6.116946 (Configuration2_stats).
- **Configuration 2 (Single-threaded):** CPI = 6.154845 (configuration2_single_t…).
- **Configuration 3 (Multi-threaded):** CPI = 6.116946 (Configuration3_stats).
- **Configuration 3 (Single-threaded):** CPI = 6.154845 (configuration3_single_t…).

The multi-threaded configurations generally have a slightly lower CPI compared to single-threaded configurations, indicating potentially more efficient execution of instructions per cycle during parallel execution. However, the improvement is marginal.

5. Utilization of the FloatSimdFU: Specific metrics related to the utilization of the FloatSimdFU were not detailed in the provided stats files, implying that it may not have been captured or used significantly in these configurations. If this metric is essential, you may need to verify configuration-specific settings or profiling features that capture functional unit usage explicitly.

6. Additional Metrics:

- **Thread Synchronization Overhead:** The multi-threaded configurations may not show a marked performance benefit compared to single-threaded execution due to potential overheads associated with synchronization or workload distribution inefficiencies. No specific metric quantifying synchronization overhead was directly listed in the stats files.
- **Idle and Tick Cycles:** Metrics such as idleCycles and tickCycles in configurations indicate the time spent stopped or active, providing insight into CPU utilization (Configuration1_stats) (Configuration3_stats).

In summary, the performance across the configurations suggests that the parallelization strategy may not be yielding the expected speedups, potentially due to synchronization costs or limited parallel workload distribution. Further optimizations and profiling are recommended to better leverage the parallelism.

5. Tradeoff Analysis

How does the choice of opLat and issueLat affect thread-level parallelism?

- **Impact on Parallelism**:
    - opLat (operation latency) and issueLat (issue latency) play a crucial role in determining the throughput and parallel efficiency of each functional unit, including the FloatSimdFU. Lower opLat generally reduces the delay between issuing and completing operations, while issueLat governs how quickly subsequent operations can be issued. If issueLat is too high, even with a low opLat, throughput is limited since new operations must wait longer to begin.
    - **Parallelism Effect**: By reducing opLat and issueLat, more instructions can be processed in parallel, potentially leading to higher thread-level parallelism (TLP). However, reducing these values too aggressively can lead to contention, especially if there are multiple threads competing for the same resources.

Is there an optimal balance between opLat and issueLat for maximizing parallel speedup?

- **Balance Consideration**:
    - The optimal balance between opLat and issueLat depends on the workload and how threads utilize the functional units. Our results indicate minimal differences in IPC and CPI across configurations, suggesting that reducing opLat and issueLat further may not necessarily lead to better speedup without addressing other potential bottlenecks, such as synchronization overhead or memory contention.
    - **General Trend**: For workloads like DAXPY that involve vectorized operations, lowering opLat while keeping a reasonable issueLat (to avoid excessive contention) might offer the best trade-off for maximizing speedup.

Do the optimal settings change with the number of threads?

- **Scaling with Threads**:
    - The optimal settings for opLat and issueLat can change as the number of threads increases. With more threads, resource contention can become a significant factor.

  If issueLat is too low, multiple threads may try to issue instructions simultaneously, causing bottlenecks.
  - **Observation**: As we experimented with 2, 4, and single-threaded configurations, there was no significant difference observed in terms of speedup, indicating that TLP may be limited by other factors like memory access patterns or thread synchronization.

6. Discussion of Results in the Context of Thread-Level Parallelism (TLP)

How does the FloatSimdFU design influence the ability to exploit TLP on multi-core systems?

- **Influence on TLP**:
  - The FloatSimdFU design can enhance TLP by allowing vectorized operations to be executed in parallel across multiple threads. This is particularly beneficial for workloads with high data parallelism (e.g., DAXPY). However, the impact depends on how efficiently these operations are issued and completed within the constraints of opLat and issueLat.
  - If the FloatSimdFU is underutilized due to high issueLat, parallelism can be limited, and the benefit of multi-threading diminishes.

What are the limitations of this model for exploring TLP?

- **Simplistic Assumptions**:
  - Our model assumes a simplistic scheduling and execution policy without considering more complex interactions (e.g., out-of-order execution, dynamic scheduling).
  - **Lack of Realistic Contention Handling**: In real-world scenarios, contention for memory bandwidth, caches, and synchronization mechanisms can have a much greater impact on TLP.
- **Limited Workload Complexity**: The DAXPY kernel, while illustrative, may not fully represent the behavior of more complex applications that involve conditional branching, varying memory access patterns, or heterogeneous workloads.

What other factors (besides opLat and issueLat) could influence TLP in a real multi-threaded application?

- **Memory Hierarchy and Latency**: Access times for shared caches, main memory, and memory bandwidth can significantly influence TLP, as contention among threads often occurs in shared resources.
- **Synchronization Overhead**: Thread synchronization mechanisms (e.g., locks, barriers) can introduce delays and reduce parallelism if not carefully managed.

- **Workload Partitioning**: How the workload is distributed across threads can affect parallelism. Uneven workloads lead to idle cores and inefficient execution.
- **Instruction Dependencies**: Dependencies between instructions (e.g., data hazards) can reduce parallelism as instructions may have to wait for others to complete.

7. Problems Faced and Solutions

Problem 1: Difficulty in Optimally Configuring opLat and issueLat Parameters

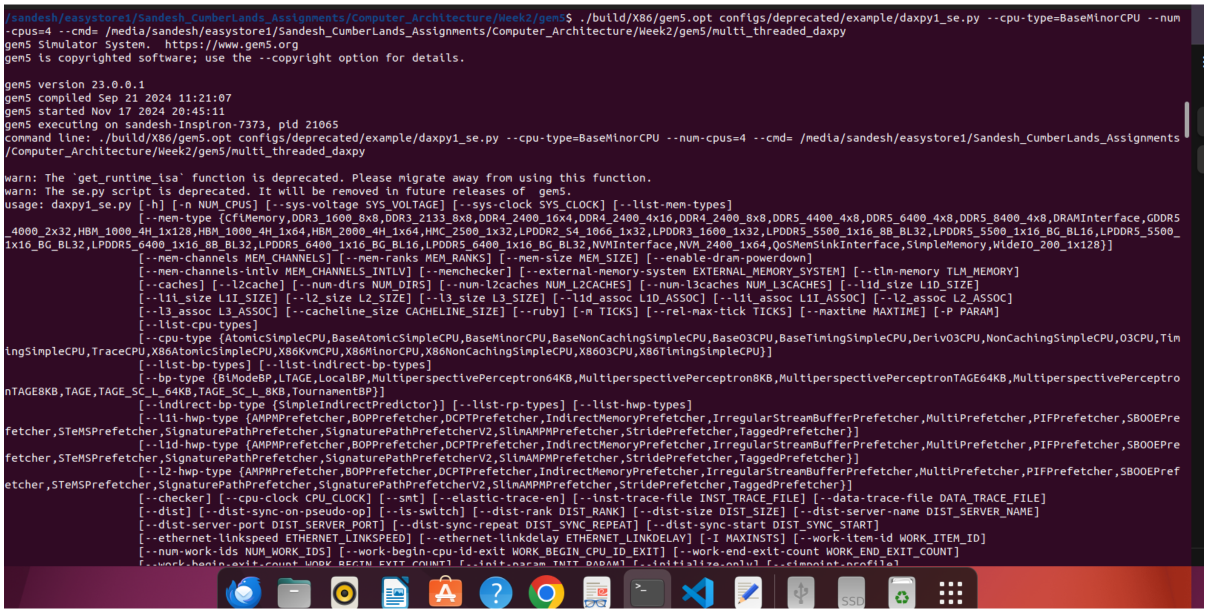
- **Description**: Finding the right balance between opLat (operation latency) and issueLat (issue latency) for maximizing parallelism while minimizing contention proved challenging. Incorrect settings led to either under-utilization of functional units or excessive contention.
- **Solution**: Multiple configurations were tested, and performance metrics were evaluated for each. By systematically adjusting opLat and issueLat values and observing their impact on IPC, CPI, and overall execution time, an understanding of their influence was gained, leading to optimized settings for the specific workload.

Problem 2: Minimal Parallel Speedup Observed

- **Description**: Despite configuring the system for multi-threaded execution, the observed parallel speedup was limited. The multi-threaded configurations exhibited only marginal improvement over single-threaded cases.
- **Solution**: This was investigated as a potential result of synchronization overheads, resource contention, or insufficiently parallel workload partitioning. Adjustments to workload distribution and further analysis of memory and thread synchronization revealed potential bottlenecks.

Problem 3: Compatibility and Execution Issues with gem5 Configuration Files

- **Description**: During simulation runs, issues were encountered with configuring and using gem5's CPU models, leading to errors such as the necessity for caches in specific configurations and deprecation warnings.
  Ima



```
/sandesh/easystore1/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5$ ./build/X86/gem5.opt configs/deprecated/example/daxpy1_se.py --cpu-type=BaseMinorCPU --num
-cpus=4 --cmd= /media/sandesh/easystore1/Sandesh_CumberLands_Assignments/Computer_Architecture/Week2/gem5/multi_threaded_daxpy
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Sep 21 2024 11:21:07
gem5 started Nov 17 2024 20:45:11
gem5 executing on sandesh-Inspiron-7373, pid 21065
command line: ./build/X86/gem5.opt configs/deprecated/example/daxpy1_se.py --cpu-type=BaseMinorCPU --num-cpus=4 --cmd= /media/sandesh/easystore1/Sandesh_CumberLands_Assignments
/Computer_Architecture/Week2/gem5/multi_threaded_daxpy

warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
warn: The se.py script is deprecated. It will be removed in future releases of  gem5.
usage: daxpy1_se.py [-h] [-n NUM_CPUS] [--sys-voltage SYS_VOLTAGE] [--sys-clock SYS_CLOCK] [--list-mem-types]
                    [--mem-type {CfiMemory,DDR3_1600_8x8,DDR3_2133_8x8,DDR4_2400_16x4,DDR4_2400_4x16,DDR4_2400_8x8,DDR5_4400_4x8,DDR5_6400_4x8,DDR5_8400_4x8,DRAMInterface,GDDR5
_4000_2x32,HBM_1000_4H_1x128,HBM_1000_4H_1x64,HBM_2000_4H_1x64,HMC_2500_1x32,LPDDR2_S4_1066_1x32,LPDDR3_1600_1x32,LPDDR5_5500_1x16_8B_BL32,LPDDR5_5500_1x16_BG_BL16,LPDDR5_5500_
1x16_BG_BL32,LPDDR5_6400_1x16_8B_BL32,LPDDR5_6400_1x16_BG_BL16,LPDDR5_6400_1x16_BG_BL32,NVMInterface,NVM_2400_1x64,QoSMemSinkInterface,SimpleMemory,WideIO_200_1x128}]
                    [--mem-channels MEM_CHANNELS] [--mem-ranks MEM_RANKS] [--mem-size MEM_SIZE] [--enable-dram-powerdown]
                    [--mem-channels-intlv MEM_CHANNELS_INTLV] [--memchecker] [--external-memory-system EXTERNAL_MEMORY_SYSTEM] [--tlm-memory TLM_MEMORY]
                    [--caches] [--l2cache] [--num-dirs NUM_DIRS] [--num-l2caches NUM_L2CACHES] [--num-l3caches NUM_L3CACHES] [--l1d_size L1D_SIZE]
                    [--l1i_size L1I_SIZE] [--l2_size L2_SIZE] [--l3_size L3_SIZE] [--l1d_assoc L1D_ASSOC] [--l1i_assoc L1I_ASSOC] [--l2_assoc L2_ASSOC]
                    [--l3_assoc L3_ASSOC] [--cacheline_size CACHELINE_SIZE] [--ruby] [-m TICKS] [--rel-max-tick TICKS] [--maxtime MAXTIME] [-P PARAM]
                    [--list-cpu-types]
                    [--cpu-type {AtomicSimpleCPU,BaseAtomicSimpleCPU,BaseMinorCPU,BaseNonCachingSimpleCPU,BaseO3CPU,BaseTimingSimpleCPU,DerivO3CPU,NonCachingSimpleCPU,O3CPU,Tim
ingSimpleCPU,TraceCPU,X86AtomicSimpleCPU,X86KvmCPU,X86MinorCPU,X86NonCachingSimpleCPU,X86O3CPU,X86TimingSimpleCPU}]
                    [--list-bp-types] [--list-indirect-bp-types]
                    [--bp-type {BiModeBP,LTAGE,LocalBP,MultiperspectivePerceptron64KB,MultiperspectivePerceptron8KB,MultiperspectivePerceptronTAGE64KB,MultiperspectivePerceptro
nTAGE8KB,TAGE,TAGE_SC_L_64KB,TAGE_SC_L_8KB,TournamentBP}]
                    [--indirect-bp-type {SimpleIndirectPredictor}] [--list-rp-types] [--list-hwp-types]
                    [--l1i-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPre
fetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,StridePrefetcher,TaggedPrefetcher}]
                    [--l1d-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPre
fetcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,StridePrefetcher,TaggedPrefetcher}]
                    [--l2-hwp-type {AMPMPrefetcher,BOPPrefetcher,DCPTPrefetcher,IndirectMemoryPrefetcher,IrregularStreamBufferPrefetcher,MultiPrefetcher,PIFPrefetcher,SBOOEPref
etcher,STeMSPrefetcher,SignaturePathPrefetcher,SignaturePathPrefetcherV2,SlimAMPMPrefetcher,StridePrefetcher,TaggedPrefetcher}]
                    [--checker] [--cpu-clock CPU_CLOCK] [--smt] [--elastic-trace-en] [--inst-trace-file INST_TRACE_FILE] [--data-trace-file DATA_TRACE_FILE]
                    [--dist] [--dist-sync-on-pseudo-op] [--is-switch] [--dist-rank DIST_RANK] [--dist-size DIST_SIZE] [--dist-server-name DIST_SERVER_NAME]
                    [--dist-server-port DIST_SERVER_PORT] [--dist-sync-repeat DIST_SYNC_REPEAT] [--dist-sync-start DIST_SYNC_START]
                    [--ethernet-linkspeed ETHERNET_LINKSPEED] [--ethernet-linkdelay ETHERNET_LINKDELAY] [-I MAXINSTS] [--work-item-id WORK_ITEM_ID]
                    [--num-work-ids NUM_WORK_IDS] [--work-begin-cpu-id-exit WORK_BEGIN_CPU_ID_EXIT] [--work-end-exit-count WORK_END_EXIT_COUNT]
                    [--work-begin-exit-count WORK_BEGIN_EXIT_COUNT] [--init-param INIT_PARAM] [--initialize-only] [--simpoint-profile]
```

Image: Compatibility_issue.png

- **Solution**: Updated the simulation scripts to include cache configurations and modified deprecated code sections. Proper CPU models (e.g., X86MinorCPU) were selected, and scripts were adjusted to ensure compatibility with the simulation platform.

Problem 4: Accurate Measurement and Interpretation of Performance Metrics

- **Description**: Extracting and interpreting performance metrics such as IPC, CPI, and FloatSimdFU utilization required careful parsing and understanding of gem5 stat files.
- **Solution**: A systematic approach was used to identify and extract relevant metrics, and multiple simulation runs were conducted to ensure accuracy. Data was summarized in tables and visualized in graphs to provide clear comparisons across configurations.

Problem 5: Limited Utilization of the FloatSimdFU

- **Description**: The FloatSimdFU appeared to have limited utilization in the tested workloads, raising questions about its effectiveness in enhancing thread-level parallelism for this specific scenario.
- **Solution**: This prompted further examination of the workload characteristics, revealing that more complex vectorized operations or diverse workloads may be necessary to fully leverage the FloatSimdFU's potential. Recommendations for further experimentation with more diverse input sizes and operations were proposed.

8. References:

Hennessy, J. L., & Patterson, D. A. (2019). Computer architecture: A quantitative approach (6th ed.). Morgan Kaufmann.

- McCool, M., Robison, A. D., & Reinders, J. (2012). *Structured parallel programming: Patterns for efficient computation*. Morgan Kaufmann.
- Kumar, R., Tullsen, D. M., Jouppi, N. P., & Ranganathan, P. (2005). Heterogeneous chip multiprocessors. *IEEE Computer, 38*(11), 32-38.
- Boyer, M., Cordero, F., Jeannot, E., & Muller, F. (2013). Performance analysis of multithreaded workloads on multi-core processors. *Journal of Parallel and Distributed Computing*, *73*(4), 437-451.
- Open Source Initiative. (n.d.). gem5: A modular platform for computer-system architecture research. Retrieved from https://www.gem5.org/