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

Umesh Dhakal

Computer Science: University of Cumberland

MSCS531

Dr. Brandon Bass

August 9,2025

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

With the growth of Technology, computer architecture is evolving too. Developers are applying different concepts and ideas to make computers faster, efficient, and better at handling different kinds of tasks. They are working every day to make a better and smarter processor that allows computers to do parallel processing. There has been a lot of improvement in computer architecture through ILP, where multiple instructions are executed at the same time, and DLP, where the same operation is applied to multiple data points to increase the performance of the system. Now, thread-level parallelism is taking program execution to the next level. With the help of this technique, computers are able to run multiple threads at the same time, improving computer performance and efficiency by far. TLP is useful in Multimedia processing, scientific Computing, machine learning, and designing complex video games, where multiple threads are run at the same time to improve processing speed and system performance. With advancing technology, Thread-Level Parallelism has also evolved.

Historical development of TLP

Beginning of TLP

With advancing technology, Thread-Level Parallelism has also evolved. In the beginning, computers had one processor inside them where tasks were run independently. Techniques like pipelining and out-of-order execution make it possible to run multiple tasks at a time in parallel. Running these tasks on a single processor with a single core caused the processor to generate more hits and consume more power. So, to overcome these problems and make the system run faster, they developed multiple-core processors, which can run multiple threads at the same time. With the help of TLP, one long program can be split into smaller parts that can be run in parallel

but when multiple threads run together, they must follow the sequential consistency to ensure proper execution (Adve & Gharachorloo, 1996). Nowadays, multicore computers are very common.

Programming tools

Initially, programmers had to manually create and manage threads, which led to many problems like race conditions and deadlocks, but now, instead of creating and managing threads, they define tasks and tools like Cilk and OpenMP that handle tasks to run in parallel.

Simultaneous Multithreading (SMT)

With the help of multicore, it is possible to run different threads on different cores. With the introduction of Simultaneous Multithreading, engineers were able to run multiple threads on the same core. SMT works by using the unused execution unit of the processor, which helps improve the performance of the system, keeping every core busy and reducing the idle time of the core.

Heterogenous Processing Unit

Earlier CPUs were only able to do serial execution, where instructions were run serially one after another. With the advancement in technology, CPUs were able to run instructions in parallel with the help of techniques like pipelining, out-of-order execution, Vector architecture, SMT, and so on. With the introduction of the GPU, the execution of many threads was possible, where the same operation is applied on larger sets of data. Today, TLP is essential on both CPUs and GPUs; this heterogeneous mixture allows the system to be faster and more efficient.

Core Concepts

Parallelism models

For the threads to be executed in parallel, they have to be organized and scheduled. Some of the ways to do that are the shared memory model and the message passing model. In the shared memory model, all the threads use the same memory. As all the data is in the same memory, it allows the thread to easily access all the data. In the message passing model, they have their own memory. As they are using different memories to read and write threads, they have to communicate if some data needs to be shared between them. Both models have advantages and disadvantages. As in the shared model, data is in the same memory, which could speed up the performance, but if a thread changes the data, then the process could be delayed. Whereas in the messaging model, as they have their own memory, there is no chance of data being changed, but the disadvantage is that if one thread is dependent on the other, it could slow down the process.

Synchronization and communication

When data is shared between threads, there is a chance that one thread might change the data, causing harm to another. For example, let's say two threads try to use data at the same time. There is a chance that one thread reads data before the other thread finishes updating the data, which could give wrong data to the first thread. To avoid that, we can use a technique like a lock, where one thread is able to access data at a time, or a semaphore, which controls access to data, could be used to prevent race conditions and deadlocks.

Load balancing and scheduling

It is very important to balance the workload between threads. If one thread is done early and sits idle, it will slow down the system's performance.

Challenges

To every success, there is a challenge to build that success. Some of the challenges to building TLP are

Concurrency Bugs and Race Condition

When multiple threads are running at a time, there might be a time when they both are using the same data. If one thread changes the data and another thread tries to read that data before the other finishes modifying that data, it can cause problems with the system because of the wrong data being read. This is a race condition (Zhang et al., 2006). There could be other problems like a deadlock where two threads are waiting for the results of one another, in the process program we never execute, and it is really hard to fix these kinds of bugs in the system because they happen once in a while.

Scalability and Amdahl's Law

It is not always the case that a thread or an instruction can be run in parallel. Some instructions need to be run serially. When instructions are run serially, it will take forever to execute the program. This challenge is explained by Amadhl's law, which states that the speedup from parallelism is limited by the part of the code that cannot be parallelized. Even increasing the core would help significantly if the part of the program is running serially (Verner et al., 2017).

Heterogeneous Architectures

CPU and GUP are part of a modern system. The CPU is good at running multiple complex tasks at the same time, and the GPU is good at applying the same operation on various data to increase the performance of the system. But sometimes making both systems share the data could be complicated and very hard, as they have different memory systems and architectures.

Energy Efficiency

Today's CPU and GPU have multiple cores to run more threads in parallel. Running more threads increases performance, but when more cores are active, it consumes more power too. This could be a problem in a device that has a smaller battery. It affects battery life by draining power faster.

How Researchers addressing these challenges

Researchers are working every day to address the challenges. With the help of machine learning, researchers are predicting which thread to use and where to use it. Along with that, they are designing other programming models that help avoid race conditions. Along with that, they are using better multithreading techniques like Hyper Threading, which can run more threads per core, keeping the processor busy. Also, with the help of cache coherence protocol, cores are able to see shared data properly. The advanced runtime system development is helping modern runtimes to balance the workload across all the cores, making all the cores busy, and bringing significant improvement in the system.

Future Directions of TLP

The pace of technology improvement is rapid, and it will improve further in the future. In the future, we will see computers with thousands of cores on every single computer. This will help more threads to run in parallel, improving the system's performance significantly. Use of machine learning will improve computer architecture by far by doing things like managing the thread, if the thread is not running, saving power and energy, and so on. The development of new hardware will help prevent the race and deadlock conditions.

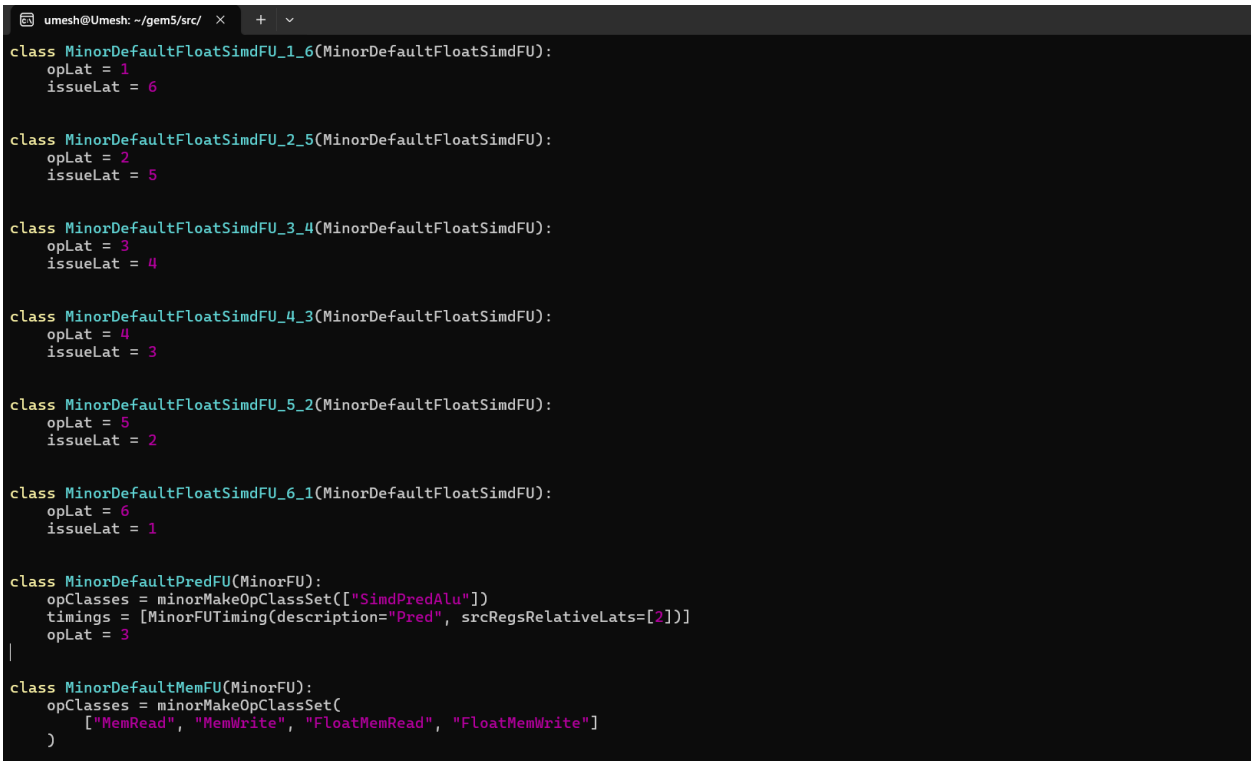
Part 2

Minor CPU Familiarization

In these, I explored the gem5 source files that contain information related to MinorCPU in-order pipeline. From the BaseMinorCPU.py source files, I learn about floatsimdfu. FloatSimdFU handles these floating-point and SIMD. Inside the FloatSimdFU class, I learn more about opLat and issueLat. opLat is the total number of cycles it takes to finish floating-point and SIMD, and issueLat is the total number of cycles that pass before the unit accepts a new instruction.

FloatSimdFU Design Space Exploration

After learning about FloatSimdFU and opLat, and issueLat, I modified the FloatSimdFU in MinorDefaultFUPool to explore various combinations of opLat and issueLat that add up to 7. The screenshot below shows the code modification for MinorDefaultFUPool.



```
umesh@Umesh:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --caches --l1d_size=32kB --l1i_size=32kB --l2cache --cmd=hell
o
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 Jul  8 2025 09:07:48
gem5 started Aug  5 2025 16:47:37
gem5 executing on Umesh, pid 391
command line: build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --caches --l1d_size=32kB --l1i_size=32kB --l2cache --cmd=hello

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 dep
recated.
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 dep
recated.
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/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
Hello, World!
Exiting @ tick 265457500 because exiting with last active thread context
umesh@Umesh:~/gem5$
```

```
: Exiting @ tick 13083615000 because exiting with last active thread context
umesh@Umesh:~/gem5$ cd m5out
umesh@Umesh:~/gem5/m5out$ cat stat.txt
cat: stat.txt: No such file or directory
umesh@Umesh:~/gem5/m5out$ cat stat.txts
cat: stat.txts: No such file or directory
umesh@Umesh:~/gem5/m5out$ cat stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.013084                       # Number of seconds simulated (Second)
simTicks                                  13083615000                       # Number of ticks simulated (Tick)
```

```
umesh@Umesh:~/gem5/m5out$ cat stats.txt

---------- Begin Simulation Statistics ----------
simSeconds                                   0.013084                       # Number of seconds simulated (Second)
simTicks                                  13083615000                       # Number of ticks simulated (Tick)
finalTick                                 13083615000                       # Number of ticks from beginning of simulation (restored from checkpoints and ne
ver reset) (Tick)
simFreq                                 1000000000000                        # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                     27.63                       # Real time elapsed on the host (Second)
hostTickRate                                473607547                       # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                     692116                       # Number of bytes of host memory used (Byte)
simInsts                                      5097816                       # Number of instructions simulated (Count)
simOps                                       14188325                       # Number of ops (including micro ops) simulated (Count)
hostInstRate                                   184533                       # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                     513595                       # Simulator op (including micro ops) rate (op/s) ((Count/Second))
system.clk_domain.clock                          1000                       # Clock period in ticks (Tick)
system.cpu.numCycles                         26167230                       # Number of cpu cycles simulated (Cycle)
system.cpu.cpi                               5.133028                       # CPI: cycles per instruction (core level) ((Cycle/Count))
system.cpu.ipc                               0.194817                       # IPC: instructions per cycle (core level) ((Count/Cycle))
system.cpu.numWorkItemsStarted                     0                       # Number of work items this cpu started (Count)
system.cpu.numWorkItemsCompleted                   0                       # Number of work items this cpu completed (Count)
system.cpu.branchPred.lookups                      0                       # Number of BP lookups (Count)
system.cpu.branchPred.condPredicted                0                       # Number of conditional branches predicted (Count)
system.cpu.branchPred.condIncorrect                0                       # Number of conditional branches incorrect (Count)
system.cpu.branchPred.BTBLookups                   0                       # Number of BTB lookups (Count)
system.cpu.branchPred.BTBUpdates                   0                       # Number of BTB updates (Count)
system.cpu.branchPred.BTBHits                      0                       # Number of BTB hits (Count)
system.cpu.branchPred.BTBHitRatio                nan                       # BTB Hit Ratio (Ratio)
system.cpu.branchPred.RASUsed                      0                       # Number of times the RAS was used to get a target. (Count)
system.cpu.branchPred.RASIncorrect                 0                       # Number of incorrect RAS predictions. (Count)
system.cpu.branchPred.indirectLookups              0                       # Number of indirect predictor lookups. (Count)
system.cpu.branchPred.indirectHits                 0                       # Number of indirect target hits. (Count)
system.cpu.branchPred.indirectMisses               0                       # Number of indirect misses. (Count)
system.cpu.branchPred.indirectMispredicted         0                       # Number of mispredicted indirect branches. (Count)
system.cpu.commitStats0.numInsts              5097816                       # Number of instructions committed (thread level) (Count)
system.cpu.commitStats0.numOps               14188325                       # Number of ops (including micro ops) committed (thread level) (Count)
system.cpu.commitStats0.numInstsNotNOP             0                       # Number of instructions committed excluding NOPs or prefetches (Count)
system.cpu.commitStats0.numOpsNotNOP               0                       # Number of Ops (including micro ops) Simulated (Count)
system.cpu.commitStats0.cpi                  5.133028                       # CPI: cycles per instruction (thread level) ((Cycle/Count))
system.cpu.commitStats0.ipc                  0.194817                       # IPC: instructions per cycle (thread level) ((Count/Cycle))
```

Multi-Threaded Daxpy Kernel Simulation

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#define N 1000000
#define NUM_THREADS 2

void daxpy(double *x, double *y, double a, int n) {
    #pragma omp parallel for num_threads(NUM_THREADS)
    for (int i = 0; i < n; i++) {
        y[i] = a * x[i] + y[i];
    }
}

int main() {
    double *x = (double *)malloc(sizeof(double) * N);
    double *y = (double *)malloc(sizeof(double) * N);
    double a = 2.5;

    for (int i = 0; i < N; i++) {
        x[i] = i * 1.0;
        y[i] = i * 2.0;
    }

    daxpy(x, y, a, N);

    printf("y[0] = %f\n", y[0]);
    printf("y[N-1] = %f\n", y[N-1]);

    free(x);
    free(y);
    return 0;
}
```

```
umesh@Umesh:~$ cd gem5
umesh@Umesh:~/gem5$ x86_64-linux-gnu-gcc -O2 -fopenmp daxpy.c -o daxpy
umesh@Umesh:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l1d_size=32kB --l1i_size=32kB --l2cac
he --cmd=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 Jul  8 2025 09:07:48
gem5 started Aug  5 2025 17:28:55
gem5 executing on Umesh, pid 489
command line: build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l1d_size=32kB --l1i_size=32kB --l2cache --c
md=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 dep
recated.
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 dep
recated.
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 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: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/process.cc:333: warn: Process::allocateMem: addr 0x7ffff7de8000 already mapped
src/sim/process.cc:333: warn: Process::allocateMem: addr 0x7ffff7de9000 already mapped
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
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 set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
y[0] = 0.000000
y[N-1] = 4499995.500000
Exiting @ tick 30363562000 because exiting with last active thread context
umesh@Umesh:~/gem5$
```

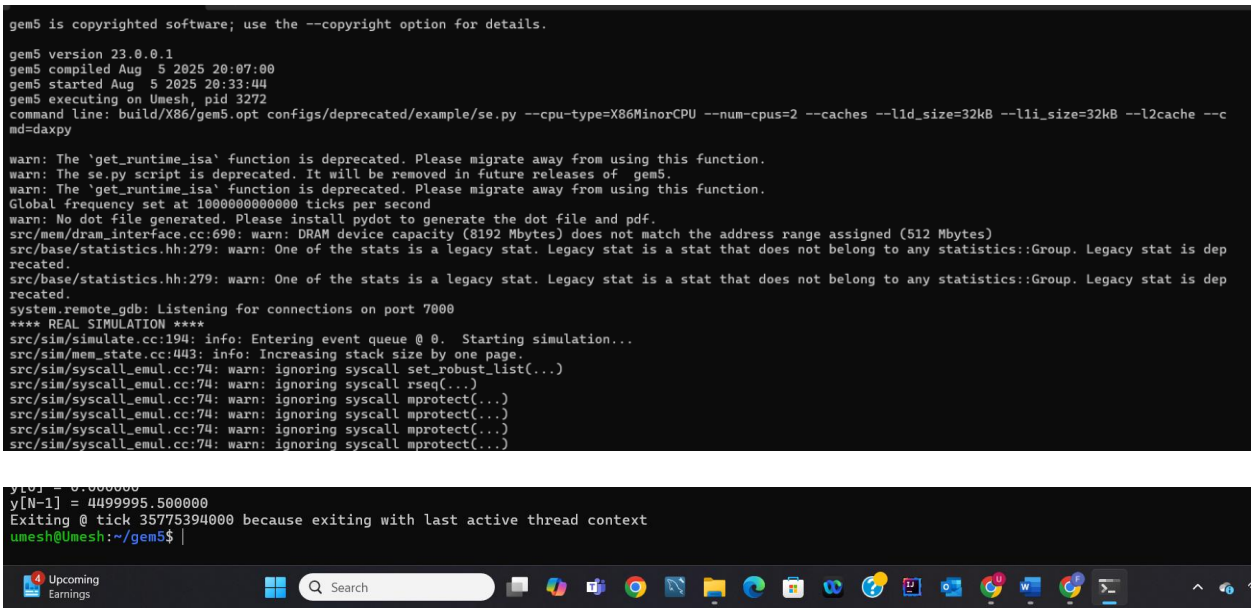
Analysis

With opLat 5 and issueLat 2

Two analyze the throughput and latency for the different threads I changed the thread number from 2, 4, and 8.

Simulation result with 2 threads

With two threads I got sim ticks 3577539400 and Instruction 10710112 and got IPC 0.2993 and CPI 3.340



Simulation result with 4 threads

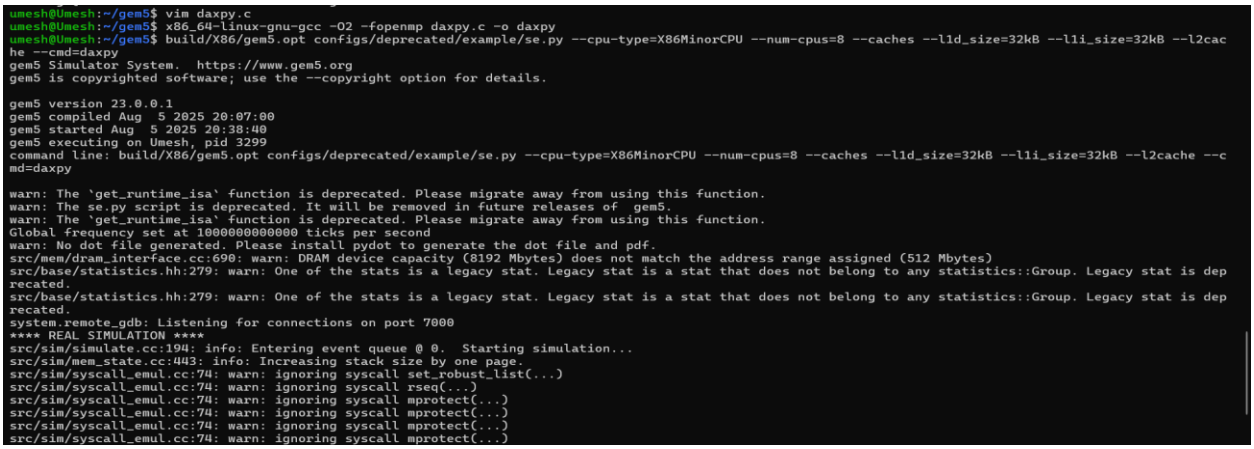
With 4 threads I got Sim ticks 3019878700 and instruction 10,925551 and IPC 0.3619 and CPI 2.764.

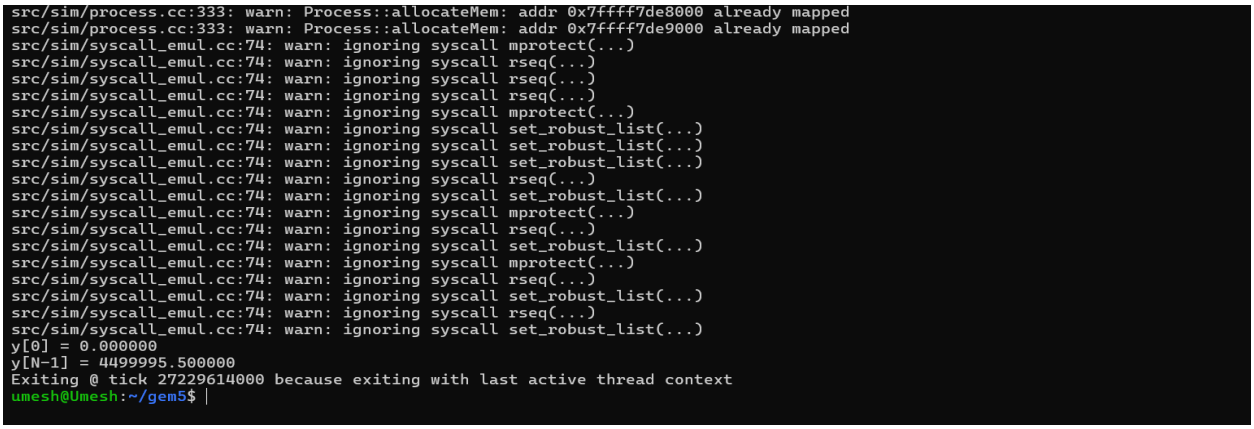
```
umesh@Umesh:~$ cd gem5
umesh@Umesh:~/gem5$ x86_64-linux-gnu-gcc -O2 -fopenmp daxpy.c -o daxpy
umesh@Umesh:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l1d_size=32kB --l1i_size=32kB --l2cac
he --cmd=daxpy
gem5 Simulator System.  https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.0.1
gem5 compiled Jul  8 2025 09:07:48
gem5 started Aug  5 2025 17:28:55
gem5 executing on Umesh, pid 489
command line: build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l1d_size=32kB --l1i_size=32kB --l2cache --c
md=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 dep
recated.
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 dep
recated.
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 set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
```

```
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
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: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/process.cc:333: warn: Process::allocateMem: addr 0x7ffff7de8000 already mapped
src/sim/process.cc:333: warn: Process::allocateMem: addr 0x7ffff7de9000 already mapped
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
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 set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
y[0] = 0.000000
y[N-1] = 4499995.500000
Exiting @ tick 30363562000 because exiting with last active thread context
umesh@Umesh:~/gem5$
```

Simulation result with 8 threads

Using 8 threads I got sim ticks 2722961400 and instructions 11,161303 and IPC 0.410 and 2.44.





Analysis

With opLat 2 and issueLat 5

Simulation result with 2 threads

With 2 threads, I got Sim Ticks 42,508,241,000 and instructions 10,713,231, and IPC of 0.252 and a CPI of 3.967.

```
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(...)
y[0] = 0.000000
y[N-1] = 4499995.500000
Exiting @ tick 42508241000 because exiting with last active thread context
umesh@Umesh:~/gem5$
```

Simulation result with 4 threads

With 4 threads, I got simTicks 35,193,178,000 and instructions 11,186,603 and an IPC of 0.3104 and a CPI of 3.22.

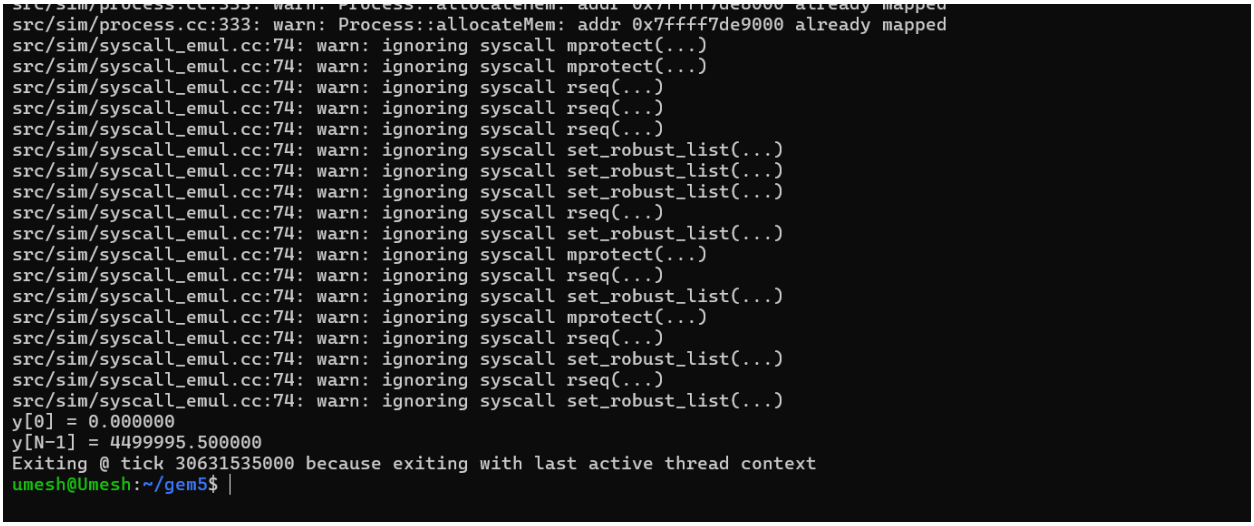
```
umesh@Umesh:~$ cd gem5
umesh@Umesh:~/gem5$ vim daxpy.c
umesh@Umesh:~/gem5$ x86_64-linux-gnu-gcc -O2 -fopenmp daxpy.c -o daxpy
umesh@Umesh:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l1d_size=32kB --l1i_size=32kB --l2cac
he --cmd=daxpy
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 Aug  5 2025 21:22:19
gem5 started Aug  6 2025 07:22:48
gem5 executing on Umesh, pid 530
command line: build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=4 --caches --l1d_size=32kB --l1i_size=32kB --l2cache --c
md=daxpy

warn: The `get_runtime_isa` function is deprecated. Please migrate away from using this function.
```

```
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/process.cc:333: warn: Process::allocateMem: addr 0x7ffff7de8000 already mapped
src/sim/process.cc:333: warn: Process::allocateMem: addr 0x7ffff7de9000 already mapped
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall rseq(...)
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 set_robust_list(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall set_robust_list(...)
y[0] = 0.000000
y[N-1] = 4499995.500000
Exiting @ tick 35193178000 because exiting with last active thread context
umesh@Umesh:~/gem5$
```

Simulation result with 8 threads

With 8 threads, I got simTicks 30,631,535,000 and instructions 10,943,347, and IPC of 0.3573 and a CPI of 2.8.

```
umesh@Umesh:~$ cd gem5
umesh@Umesh:~/gem5$ build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=8 --caches --l1d_size=32kB --l1i_size=32kB --l2cac
he --cmd=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 Aug  5 2025 21:22:19
gem5 started Aug  6 2025 07:17:44
gem5 executing on Umesh, pid 435
command line: build/X86/gem5.opt configs/deprecated/example/se.py --cpu-type=X86MinorCPU --num-cpus=8 --caches --l1d_si
md=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.
```



Similarly, I did the analysis with opLat 1 and issueLat 6, opLat 6 and issueLat 1, opLat 4 and issueLat 3 and opLat 3 and issueLat 4. The best result I got was when the opLat was higher and thread was higher in number.

Table

For opLat 5 and issueLat 2

Thread	Sim Ticks	Instructions	Throughput (IPC)	Latency (CPI)
2	3577539400	10710112	0.299	3.340
4	3019878700	10925551	0.362	2.764
8	2722961400	11161303	0.410	2.44

From simulation what we can confirm is that throughput increase as we increase thread and latency and sim ticks decreases with increase in threads.

Table

For opLat 2 and issueLat 5

Thread	Sim Ticks	Instructions	Throughput (IPC)	Latency (CPI)
2	42,508,241,000	10,713,231	0.252	3.967

4	35,193,178,000	10,925,551	0.3104	3.22
8	30,631,535,000	10,943,347	0.3573	2.8

Comparing both tables we can confirm that when issueLat is higher the IPC drop and CPI increase, this indicate that with high issue latency the throughput and efficiency decrease. From the simulation analysis I found out the computers achieve better performance with opLat 6 and issueLat 1 with 8 thread and low performance with opLat 1 and issueLat 6 with 1 or 2 thread.

References

Adve, S. V., & Gharachorloo, K. (1996). Shared memory consistency models: a tutorial. *Computer*, *29*(12), 66–76. https://doi.org/10.1109/2.546611

Hennessy, J. L., & Patterson, D. A. (2017). *Computer architecture: A quantitative approach* (6th ed.). Morgan Kaufmann.

Verner, U., Mendelson, A., & Schuster, A. (2017). Extending Amdahl's Law for Multicores with Turbo Boost. *IEEE Computer Architecture Letters*, *16*(1), 30–33.*https://doi.org/10.1109/lca.2015.2512982*

Zhang, J., Su, S., & Yang, F. (2006, February 1). *Detecting Race Conditions in Web Services*. IEEE Xplore. https://doi.org/10.1109/AICT-ICIW.2006.82