Accelerating Simulations in gem5

A presentation by

Kaustav Goswami

UC DAVIS
COMPUTER SCIENCE

Outline of this session

- We will first address **<u>why</u>** do we need to make simulations faster, and, then we will see **<u>how</u>** can we make simulations faster.
- We will work together to create a *runscript,* that we will use throughout this session.
- We will iterate over each of the techniques of accelerating simulations, and work on several hands-on sessions.
- We will keep on modifying the *runscript.*

```
> cd materials/using-gem5/09-accelerating-simulations/
```

gem5

Hands-on Session: Matrix Multiply

```
kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$ g++ mm_base.cpp
kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$ ./a.out
Printing Statistics :: Wall Clock Time
======================================
Program: 4.7777e-05 s
Matrix Multiply: 4.191e-06 s

kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$ g++ mm_base.cpp
kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$ ./a.out
Printing Statistics :: Wall Clock Time
======================================
Program: 0.00500679 s
Matrix Multiply: 0.00389788 s

kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$ g++ mm_base.cpp
kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$ ./a.out
Printing Statistics :: Wall Clock Time
======================================
Program: 9.66581 s
Matrix Multiply: 9.53323 s

kaustavg@citra:/scr/kaustavg/projects/bootcamp-code$
```

```
> cd data-files/base
> g++ mm_base.cpp
```

- Go to the ***data-files*** directory.
- Compile the matrix multiply **(mm_base.cpp)** program on your host machine.
- Run the program with 100x100 matrix.
- Increase the size to 1000x1000.

What's wrong?

gem5

How do we make gem5 simulations faster?

Annotating workloads

Using Checkpoints

Sampling

Creating the pre-requisite runscript

- We need to write a configuration script to execute the *mm* code.
- We will use the **SimpleBoard** in gem5 to do SE mode simulations.
- This will be a work along session.
- There are three **TODO**s marked in the runscript.
- Run the binary file with 100x100 elements.

```
> cd 00-prerequisite-runscript
```

gem5

Rationale Behind Accelerating Simulations

- We need a mechanism in gem5 for the simulated program can **communicate** with the host machine.
- This allows us to **manipulate** the runscript.
- We can keep only **essential statistics** of the simulation.

gem5 EXIT Events

- EXIT events in gem5 allows us to **communicate** from the simulator to the host.
- These are implemented as ***magic instructions*** or ***memory mapped I/O***.
- There are different types of **EXIT Events** in gem5.

Different types of EXIT Events in gem5

- ExitEvent.**EXIT**
- ExitEvent.**CHECKPOINT**
- ExitEvent.**FAIL**
- ExitEvent.**SWITCHCPU**
- ExitEvent.**WORKBEGIN**
- ExitEvent.**WORKEND**
- ExitEvent.**USER_INTERRUPT**
- ExitEvent.**MAX_TICK**

Annotations

- We only care about **important regions** during the simulation.
- In other words, we have **regions of interests** or **ROIs**.
- How do we make gem5 **understand** ROIs?

The m5 utility!

The *m5* instruction

- What is **m5**?
- m5 can be implemented as a **magic instruction** or as **memory-mapped I/O** or **MMIO**.
- We add m5 library calls in the source code.

m5 Library Calls

- We need to **identify** ROIs.
- Look at the previous example, but with annotations.
- We use **m5_work_begin(M, N)** at the beginning and **m5_work_end(M, N)** at the end of the ROI.
- This resets the statistics which is recorded in the **m5out/stats.txt** file.

```
    // Naive matrix multiplication code. It performs N^3 computations. We also
    // keep a track of time for this part of the code.
    //
    auto mm_start = std :: chrono :: high_resolution_clock :: now();

    // annotating the ROI
    //
#ifdef GEM5
    m5_work_begin(0, 0);
#endif

    for(int i = 0 ; i < N ; i++)
        for(int j = 0 ; j < N ; j++)
            for(int k = 0 ; k < N ; k++)
                C[i][j] += A[i][k] * B[k][j];

    // end of ROI
    //
#ifdef GEM5
    m5_work_end(0);
#endif

    auto mm_end = std :: chrono :: high_resolution_clock :: now();

    // Free the memory allocated.
    //
    delete data_A;
    delete data_B;
    delete data_C;
    delete A;
    delete B;
    delete C;
```

gem5

Annotation Example

- **Go to** data_files/annotated
- **Open and review** the mm_annotated.cpp file.
- **Compile** the mm_annotated.cpp

Wait! How do we compile the code?

gem5

Recapping how to compile annotated code

- The m5 utility must be **compiled** in the util/m5.
- Change the **GEM5_HOME** variable.
- We need to **modify** the CFLAGS and LDFLAGS in the *Makefile*.
- We need to **include** the **m5ops.h** file from *include/gem5/*directory.
- **Make**

```
> cd gem5
```

```
➜  m5 git:(stable) ✗ scons build/x86/out/m5
scons: Reading SConscript files ...
Checking for java package org.junit...(cached) no
junit test framework not found, not build java wrapper test
Checking whether pkg-config program exists.../usr/bin/pkg-config
Checking for pkg-config package lua51...(cached) no
lua 5.1 not detected, not building lua wrapper.
scons: done reading SConscript files.
scons: Building targets ...
```

```
#CFLAGS

CFLAGS=-std=c++14 -static -O3 -I$(GEM5_HOME)/include

#LDFLAGS

LDFLAGS=-L$(GEM5_HOME)/util/m5/build/$(TARGET_ISA)/out -lm5

```

```

#define GEM5

#ifdef GEM5
#include <gem5/m5ops.h>
#endif

```

Hands-on Session I: Modify the runscript

- Modify the original runscript to execute the *mm_annotated* binary.
- Print the total number of ticks.
- Print the EXIT Event.
- Look at the number of ticks in the m5out/stats.txt file.

```
# The `simulator` module has these methods which can be helpful for this
# hands-on session:
#
# simulator.get_current_tick()          -> returns the current tick
# simulator.get_last_exit_event_cause() -> returns the last EXIT event cause
```

```
> cd 01-modifying-runscript
```

Some m5 function calls in gem5

- **m5_reset(M, N)**: It resets the stats file. A new section is generated in the m5out/stats.txt file.
- **m5_dump_reset_stats(M, N):** It dumps the stats and resets the stats file.
- **m5_work_begin(X, Y):** It starts keeping stats.
- **m5_work_end(X, Y)**: It stops keeping stats.
- **m5_exit(M)**: It drops a ExitEvent.EXIT EXIT Event in the runscript.

```
> cd gem5/include/gem5
> code m5ops.h
```

gem5

Understanding the different m5 function calls

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.173081                       # Number of seconds simulated (Second)
simTicks                                     173081275137                   # Number of ticks simulated (Tick)
finalTick                                    173081275137                   # 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                                  8.03                           # Real time elapsed on the host (Second)
hostTickRate                                 21564253748                    # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                   1225464                        # Number of bytes of host memory used (Byte)
simInsts                                     2386715                        # Number of instructions simulated (Count)
simOps                                       4150319                        # Number of ops (including micro ops) simulated (Count)
hostInstRate                                 297357                         # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                   517082                         # Simulator op (including micro ops) rate (op/s) ((Count/Second))
```

Fig. Executing without ROI annotations.

```
---------- Begin Simulation Statistics ----------
simSeconds                                   0.008159                       # Number of seconds simulated (Second)
simTicks                                     8159148018                     # Number of ticks simulated (Tick)
finalTick                                    8159148018                     # 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                                  0.41                           # Real time elapsed on the host (Second)
hostTickRate                                 19712732539                    # The number of ticks simulated per host second (ticks/s) ((Tick/Second))
hostMemory                                   1180944                        # Number of bytes of host memory used (Byte)
simInsts                                     111968                         # Number of instructions simulated (Count)
simOps                                       209867                         # Number of ops (including micro ops) simulated (Count)
hostInstRate                                 270466                         # Simulator instruction rate (inst/s) ((Count/Second))
hostOpRate                                   506938                         # Simulator op (including micro ops) rate (op/s) ((Count/Second))
```

Fig. Executing with ROI begin and end.

Understanding the different m5 function calls

```

---------- Begin Simulation Statistics ----------
simSeconds                                   0.000956                       # Number of seconds simulated (Second)
simTicks                                    955552491                       # Number of ticks simulated (Tick)
finalTick                                  8161572591                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
```

Fig. Executing with *m5_reset_stats(0, 0)*.

```

---------- Begin Simulation Statistics ----------
simSeconds                                   0.006500                       # Number of seconds simulated (Second)
simTicks                                   6499623204                       # Number of ticks simulated (Tick)
finalTick                                  6499623204                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
```

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

---------- Begin Simulation Statistics ----------
simSeconds                                   0.000706                       # Number of seconds simulated (Second)
simTicks                                    706311315                       # Number of ticks simulated (Tick)
finalTick                                  7205934519                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
```

```

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

---------- Begin Simulation Statistics ----------
simSeconds                                   0.000955                       # Number of seconds simulated (Second)
simTicks                                    955384326                       # Number of ticks simulated (Tick)
finalTick                                  8161318845                       # Number of ticks from beginning of simulation (restored from checkpoints and never reset) (Tick)
```

Fig. Executing with *m5_dump_reset_stats(0, 0)*.

Annotating a real-world workload.

- As computer architects, we must work with standard benchmark programs to **analyze the performance and requirements** of the proposed design.
- It is important to understand the **characteristics** of the workload.
- In this session, we will investigate the **LLVM's Stanford Benchmark Suite** [1].

[1] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis & transformation," International Symposium on Code Generation and Optimization, 2004. CGO 2004., 2004, pp. 75-86, doi: 10.1109/CGO.2004.1281665.

Hands-on Session II: Annotating real-world workload.

- We will annotate *Bubblesort* in this session.
- Download the benchmarks.
- Edit the Bubblesort.cpp file.
- Compile
- Simulate

```
> cd 02-annotating-llvm-stanford
```

```
void Bubble(int run) {
    int i, j;
    bInitarr();
    top=srtelements;

    while ( top>1 ) {

        i=1;
        while ( i<top ) {

            if ( sortlist[i] > sortlist[i+1] ) {
                j = sortlist[i];
                sortlist[i] = sortlist[i+1];
                sortlist[i+1] = j;
            }
            i=i+1;
        }

        top=top-1;
    }
    if ( (sortlist[1] != littlest) || (sortlist[srtelements] != biggest) )
    printf ( "Error3 in Bubble.\n");
    printf("\n%d\n", sortlist[run + 1]);

}

int main()
{
    int i;
    for (i = 0; i < 10; i++) Bubble(i);
    return 0;
}
```

gem5

Analyzing Hands-on Session II

- Are there repeating patterns? The entire *Bubble(i)* function is being called for 10 times.
- Do we need the same stats 10 times?
- Let us use **m5_exit(M)** at the end.

```
> cd 02-annotating-llvm-stanford/optimized
```

```
void Bubble(int run) {
    int i, j;
    bInitarr();
    top=srtelements;
#ifdef GEM5
    m5_work_begin(0, 0);
#endif
    while ( top>1 ) {

        i=1;
        while ( i<top ) {

            if ( sortlist[i] > sortlist[i+1] ) {
                j = sortlist[i];
                sortlist[i] = sortlist[i+1];
                sortlist[i+1] = j;
            }
            i=i+1;
        }

        top=top-1;
    }
    if ( (sortlist[1] != littlest) || (sortlist[srtelements] != biggest) )
    printf ( "Error3 in Bubble.\n");
    printf("\n%d\n", sortlist[run + 1]);
#ifdef GEM5
    m5_work_end(0, 0);
#endif

}

int main()
{
    int i;
    for (i = 0; i < 10; i++) Bubble(i);
    return 0;
}
```

More on workload annotation

- Refer to previously annotated benchmark programs such as NAS-Parallel-Benchmarks [2], parsec [3] and gabps [4] for understanding more on workload annotation.

[2] Bailey, David H., et al. "The NAS parallel benchmarks summary and preliminary results." Supercomputing'91: Proceedings of the 1991 ACM/IEEE conference on Supercomputing. IEEE, 1991.
[3] Christian Bienia *et al.* 2008. The PARSEC benchmark suite: characterization and architectural implications. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08). ACM, New York, NY, USA, 72-81. https://doi.org/10.1145/1454115.1454128
[4] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," 2015. [Online]. Available: https://arxiv.org/abs/1508.03619

gem5

```
> echo “Restoring the session: Accelerating Simulations”
```

gem5

Housekeeping

- There was a mistake in the m5_work_begin() and m5_work_end() slides/codes.
- **Thanks for pointing this out.**
- The set_se_workload **never exits** when encounters a work item.
- We must **explicitly** set `board.exit_on_work_items = True` *(or wait for the next patch).*

```python
# Set whether to exit on work items for the se_workload
board.exit_on_work_items = True
```

```
build/X86/sim/simulate.cc:194: info: Entering event queue @ 1225239867.  Starting simulation...
build/X86/sim/simulate.cc:194: info: Entering event queue @ 7273090629.  Starting simulation...
Printing Statistics :: Wall Clock Time
======================================
build/X86/sim/mem_state.cc:443: info: Increasing stack size by one page.
Program: 0.00723091 s
Matrix Multiply: 0.00604789 s

Exiting @ tick 7285127580 because exiting with last active thread context.
➜ gem5 git:(stable) ✗
```

```

---------- Begin Simulation Statistics ----------
simSeconds                                   0.006048                       # Number of seconds simulated (Second)
simTicks                                   6047850762                       # Number of ticks simulated (Tick)
finalTick                                  7273087632                       # Number of ticks from beginning of simulation (restored from checkpoint
simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
hostSeconds                                      5.53                       # Real time elapsed on the host (Second)
hostTickRate                               1093265275                       # The number of ticks simulated per host second (ticks/s) ((Tick/Second)
hostMemory                                    1185672                       # Number of bytes of host memory used (Byte)
```

```
system.workload.inst.arm                            0                       # number of arm instructions executed (Count)
system.workload.inst.quiesce                        0                       # number of quiesce instructions executed (Count)

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

---------- Begin Simulation Statistics ----------
simSeconds                                   0.000012                       # Number of seconds simulated (Second)
simTicks                                     11962359                       # Number of ticks simulated (Tick)
finalTick                                  7285049991                       # Number of ticks from beginning of simulation (restored from checkpoint
simFreq                                 1000000000000                       # The number of ticks per simulated second ((Tick/Second))
```

```python
from gem5.simulate.exit_event_generators import default_exit_generator
```

```python

simulator = Simulator(board=board, on_exit_event = {
    ExitEvent.WORKEND : default_exit_generator()
    }
)
```

Checkpoints in gem5

- Simulations take a **long time**.
- We need to be able to **save** and **restore** the **state of simulation** in gem5.
- Checkpoints were originally designed to be used with **AtomicSimpleCPU.**
- We can take checkpoints **whenever** we want.
- We will use the EXIT Event: **ExitEvent.CHECKPOINT**.
- It can be triggered by the function **m5_checkpoint(M, N).**

Checkpoints in gem5

- The *cpt* file stores the simulation values of all the SimObjects.
- What happens when we want to switch components?
- What happens to the cache when you load a checkpoint?

```
## checkpoint generated: Thu Dec 30 09:24:09 2021

[board.processor.cores.core]
instCnt=1
_pid=4294967295

[board.processor.cores.core.xc.0]
regs=0 327700 0 0 0 0 0 4294967295 255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
pendingSmi=false
smiVector=0
pendingNmi=false
nmiVector=0
pendingExtInt=false
extIntVector=0
pendingInit=false
initVector=0
pendingStartup=false
startupVector=0
startedUp=false
pendingUnmaskableInt=false
pendingIPIs=0
IRRV=0
ISRV=0
apicTimerEventScheduled=false
apicTimerEventTick=0
_status=0
floatRegs.i=0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
vecRegs=
vecPredRegs=
intRegs=0 0 0 0 140737488350656 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ccRegs=0 0 0 0 0
_pc=4214660
_upc=0
_npc=4214668
_nupc=1
_size=0

[board.processor.cores.core.workload]
brkPoint=6135808
stackBase=140737488351232
stackSize=4096
maxStackSize=8388608
stackMin=140737488347136
nextThreadStackBase=140737479962624
mmapEnd=140737354133504

[board.processor.cores.core.workload.vmalist]
size=2
```

gem5

Checkpointing the mm code

- Let us checkpoint the simulation before the matrix multiplication starts.
- We need to add an ***m5_checkpoint(0, 0)***
- The *simulator* module can easily restore the saved checkpoint.
- We also need to modify the runscript.

```
#ifdef GEM5
    m5_checkpoint(0, 0);
#endif

    for(int i = 0 ; i < N ; i++)
        for(int j = 0 ; j < N ; j++)
            for(int k = 0 ; k < N ; k++)
                C[i][j] += A[i][k] * B[k][j];

    // end of ROI
    //
```

```
# restoring Previously Stored checkpoint.

from pathlib import Path
simulator = Simulator(board=board, checkpoint_path=Path("checkpoint-dir"))
```

```

# Creating the checkpoint directory.

if not os.path.exists(os.path.join(os.getcwd(), "checkpoint-dir")):
    os.makedirs(os.path.join(os.getcwd(), "checkpoint-dir"))

# Saving the checkpoint.

simulator.save_checkpoint("checkpoint-dir")
```

More on Checkpoints

- Checkpoints can be taken after a certain number of ticks have been simulated.
- Checkpoints can also be taken periodically.
- On can also trigger ExitEvent.USER_INTERRUPT and write a checkpoint to continue the simulation later.

```python

simulator = Simulator(board=board)

simulator.run(max_ticks = 10000)

print(
    "Exiting @ tick {} because {}.".format(
        simulator.get_current_tick(),
        simulator.get_last_exit_event_cause(),
    )
)

simulator.save_checkpoint("checkpoint-dir")

```

```python
for i in range(10):
    simulator.run(max_ticks = 1000000)

    print(
        "Exiting @ tick {} because {}.".format(
            simulator.get_current_tick(),
            simulator.get_last_exit_event_cause(),
        )
    )

    simulator.run("checkpoint-dir")

```

Hands-on Session III: Restoring a checkpoint

- In this assignment, you will be provided with a **cpt** file of the mm code.
- This checkpoint file was taken after executing 1T ticks on the machine defined by the runscript.
- Your objective will be to:
  - Restore the checkpoint.
  - Execute the next 10B ticks.
  - Save the checkpoint.

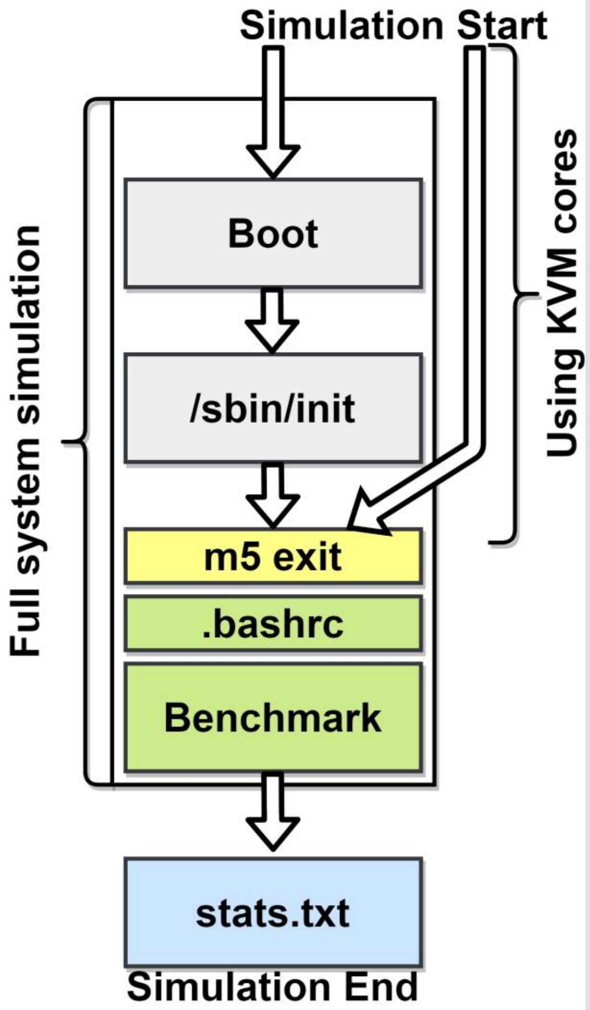
```
> cd 03-checkpoints
```

```python
# restoring Previously Stored checkpoint.

from pathlib import Path
simulator = Simulator(board=board, checkpoint_path=Path("checkpoint-dir"))

```

```python
for i in range(10):
    simulator.run(max_ticks = 1000000)

    print(
        "Exiting @ tick {} because {}.".format(
            simulator.get_current_tick(),
            simulator.get_last_exit_event_cause(),
        )
    )

    simulator.run("checkpoint-dir")

```

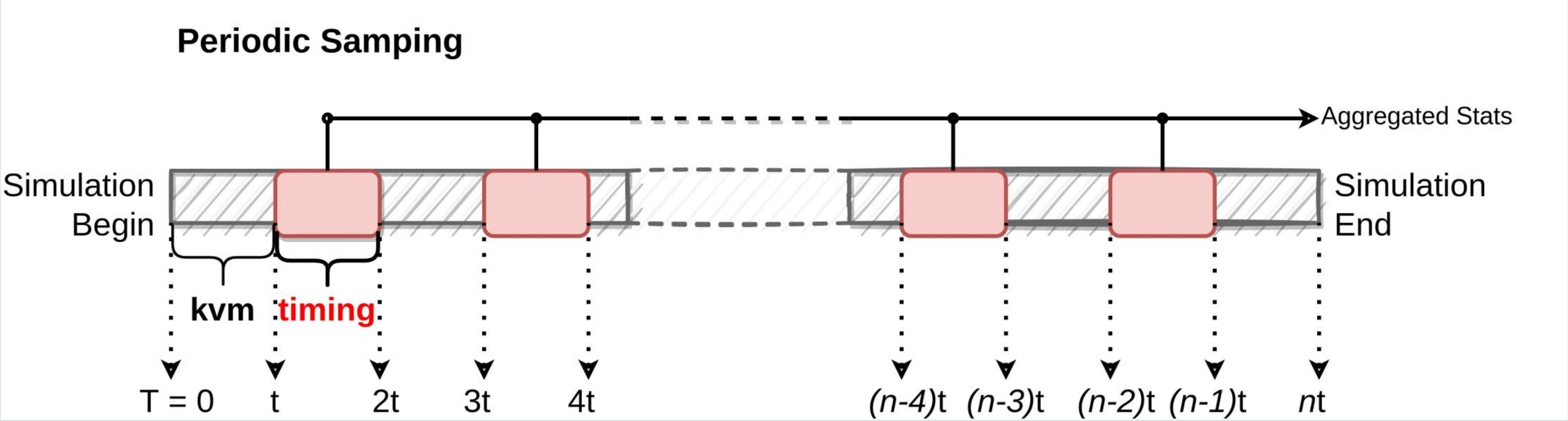
Fast-forwarding

- Once we have our **code annotated**, we ***fastforward*** our simulations.
- gem5 supports **switchable CPUs**.
- We use **kvm** or **atomic cpu** to simulate the non-essential regions of the code. Then we switch to any ***timing cpu***.



Sampling

- We can sample ***essential*** parts of the simulation to find a ***representative statistics*** for the whole workload.
- We use a combination of both *kvm/atomic* and *timing* CPUs.



[*] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder. 2003. Using SimPoint for accurate and efficient simulation. SIGMETRICS Perform. Eval. Rev. 31, 1 (June 2003), 318–319. https://doi.org/10.1145/885651.781076

gem5

Thank You!