

Using OMPTools for Scaling Full-system Simulation of ARM SVE Processors

Matthew Baker

Jeffrey Young

Oscar Hernandez

September 23, 2020

Oak Ridge National Laboratory

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

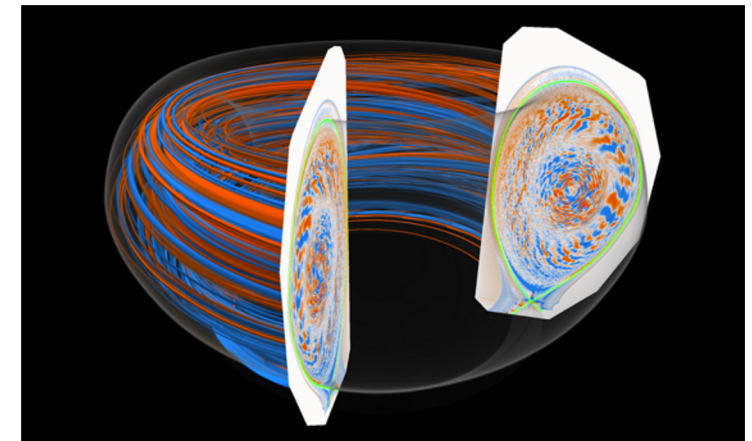
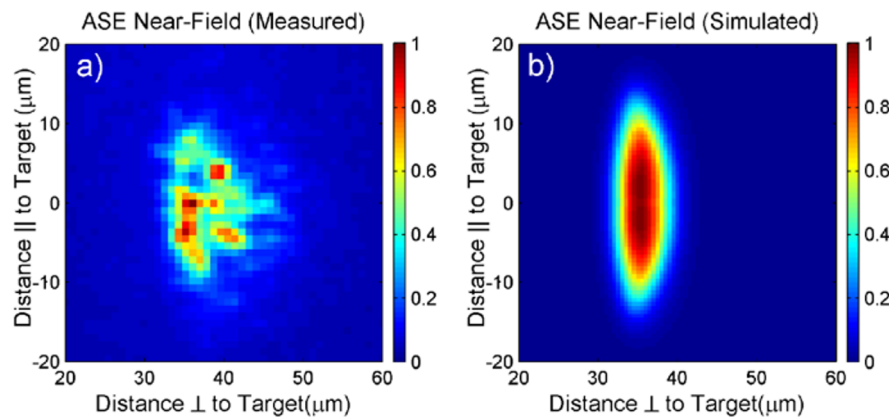
This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725



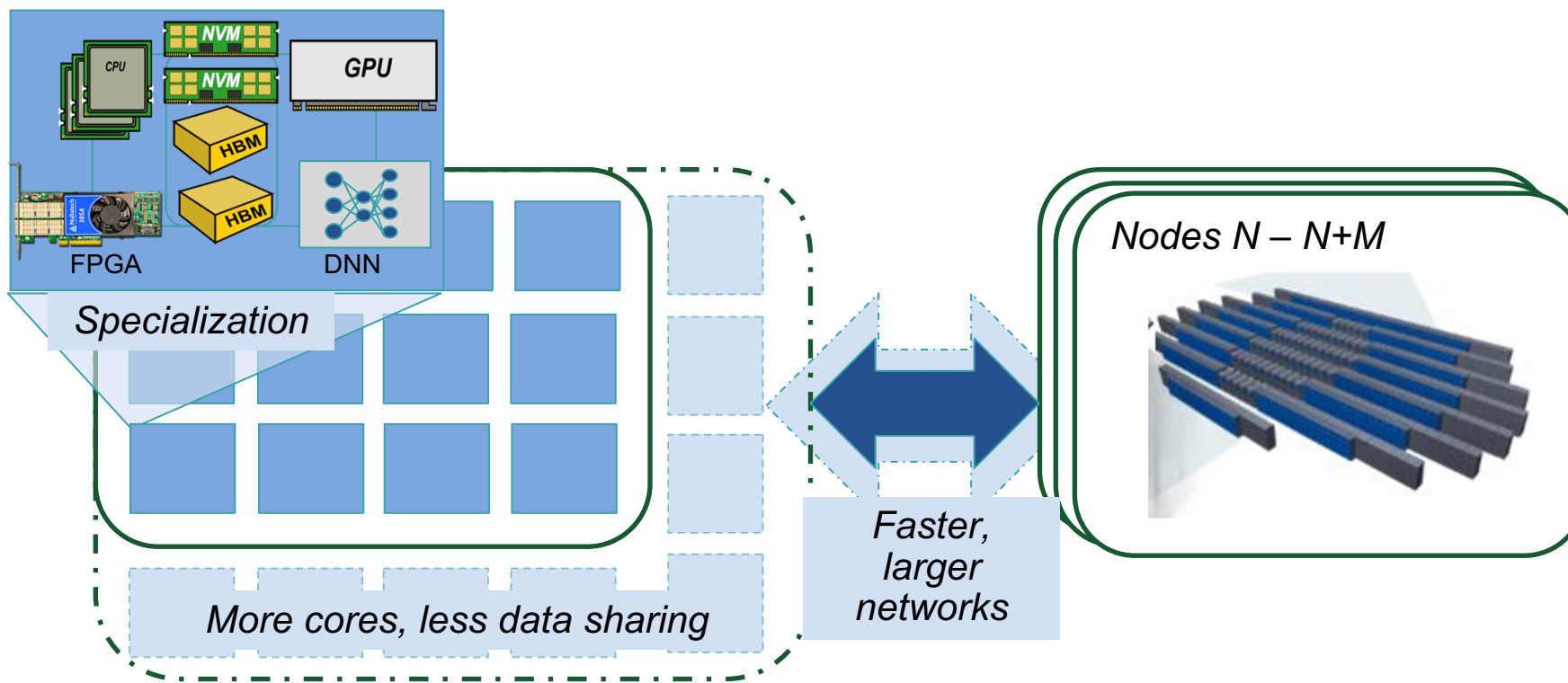
U.S. DEPARTMENT OF
ENERGY

Improving Co-design of Software and Hardware

- Software
 - The DoE uses mini-apps to represent workloads of interest, many using OpenMP, CUDA, HIP.
- Hardware
 - Can we use simulation to model future hardware and architectures at scale?
- We also want to investigate how OpenMP mini-apps might use innovative ISA extensions such as ARM's SVE
 - XRayTrace, XGC, EPCC, etc.



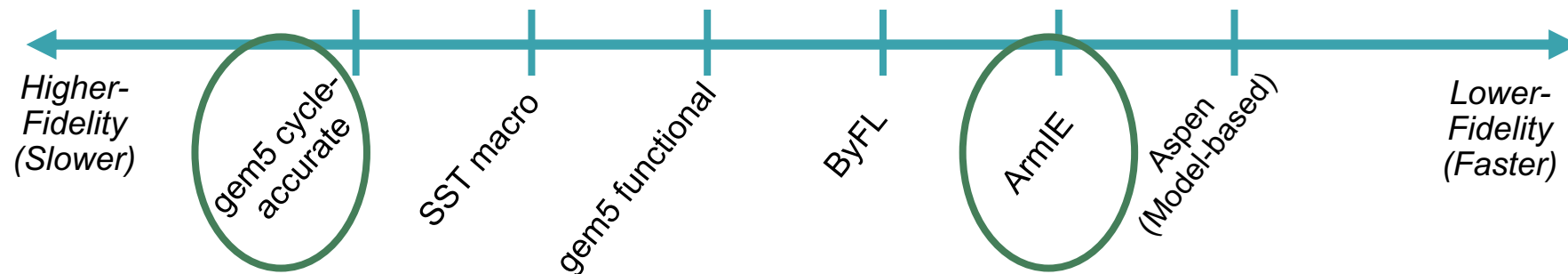
What do we mean by future architectures?



- Outside of quantum, neuromorphic, etc., future architectures will evolve into “extreme” versions of today’s systems
 - 3D stacked processors, less cache, more on-die memory, more specialization, optical interconnects
 - **Vendor input is needed to truly simulate next-gen processors**
- We plan to create simulation configurations to help answer these questions:
 1. **Should we continue to create larger nodes and dedicate transistors to cores and further specialization to handle evolving workloads?**
 2. **Should we instead focus on interconnects and data movement?**

Post Exascale

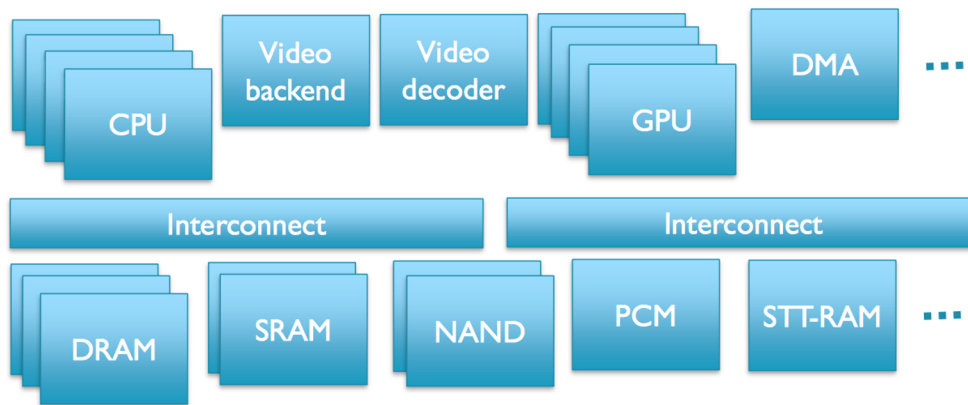
- ORNL's core capability is to "Scale computing and data analytics to post Exascale and beyond for science and energy"
 - Support mission to develop the science and technology to take full advantages of new HPC machines and be ready for the next phase – Beyond Moore's Law
- Architecture analysis typically trades off accuracy with speed
 - Full-system simulators can provide detailed models for future architectures (cycle accurate – e.g. cache misses, threads interaction, out-of-order execution, mem. interconnects)
 - Analytical model-based approaches (e.g., PALM, ByFL, Aspen, ArmIE) allow for fast evaluation of a large design space but typically lack details for coherence models, software interactions, and network.



gem5 Simulator

- **We use gem 5 to experiment with and evaluate future architectures**
 - Full-system simulation (FS) provides the best platform for co-designing programming model extensions with future hardware configurations
 - *gem5 provides the best combination of **high-fidelity, vendor-supported models** (ARM, AMD, etc.) with techniques to reduce traditional simulation overheads*
 - *Configurable using Python scripts that call C++ class implementations*
 - *Ability to simulate heterogeneous nodes with different components (e.g. GPUs, High-Bandwidth Memories (HBM))*

Heterogeneous system Simulation (gem5)



Parameters Selection

```
import m5
```

```
class L1DCache(m5.objects.Cache):  
    assoc = 2  
    size = '16kB'
```

```
class L1ICache(L1DCache):  
    assoc = 16
```

```
l1i = L1ICache(assoc=8,  
              repl=m5.objects.RandomRepl())
```

Output of gem5 when simulating an architecture

```
----- Begin Simulation Statistics -----
seconds          5.124245          # Number of seconds simulated
ticks            5124245373500      # Number of ticks simulated
al_tick          5124245373500      # Number of ticks from beginning of simulation (restored from checkpoints and never reset)
freq             1000000000000      # Frequency of simulated ticks
t_inst_rate      2033365            # Simulator instruction rate (inst/s)
t_op_rate        4128137            # Simulator op (including micro ops) rate (op/s)
t_tick_rate      49326806757        # Simulator tick rate (ticks/s)
t_mem_usage      1002472            # Number of bytes of host memory used
t_seconds        103.88             # Real time elapsed on the host
t_insts          211233255          # Number of instructions simulated
t_ops            428845682          # Number of ops (including micro ops) simulated
tem.voltage_domain.voltage 1        # Voltage in Volts
tem.clk_domain.clock 1000          # Clock period in ticks
tem.mem_ctrls.pwrStateResidencyTicks::UNDEFINED 5124245373500 # Cumulative time (in ticks) in various power states
tem.mem_ctrls.bytes_read::cpu.dtb.walker 196424      # Number of bytes read from this memory
tem.mem_ctrls.bytes_read::cpu.itb.walker 104936      # Number of bytes read from this memory
tem.mem_ctrls.bytes_read::cpu.inst 2076302352        # Number of bytes read from this memory
tem.mem_ctrls.bytes_read::cpu.data 89576607          # Number of bytes read from this memory
tem.mem_ctrls.bytes_read::pc.south_bridge.ide 37264    # Number of bytes read from this memory
tem.mem_ctrls.bytes_read::total 2166217583          # Number of bytes read from this memory
tem.mem_ctrls.bytes_inst_read::cpu.inst 2076302352    # Number of instructions bytes read from this memory
tem.mem_ctrls.bytes_inst_read::total 2076302352      # Number of instructions bytes read from this memory
tem.mem_ctrls.bytes_written::cpu.itb.walker 16        # Number of bytes written to this memory
tem.mem_ctrls.bytes_written::cpu.data 72221164       # Number of bytes written to this memory
tem.mem_ctrls.bytes_written::pc.south_bridge.ide 2990080 # Number of bytes written to this memory
tem.mem_ctrls.bytes_written::total 75211260         # Number of bytes written to this memory
tem.mem_ctrls.num_reads::cpu.dtb.walker 24553        # Number of read requests responded to by this memory
tem.mem_ctrls.num_reads::cpu.itb.walker 13117        # Number of read requests responded to by this memory
tem.mem_ctrls.num_reads::cpu.inst 259537794         # Number of read requests responded to by this memory
tem.mem_ctrls.num_reads::cpu.data 15412406          # Number of read requests responded to by this memory
tem.mem_ctrls.num_reads::pc.south_bridge.ide 850      # Number of read requests responded to by this memory
tem.mem_ctrls.num_reads::total 274988720            # Number of read requests responded to by this memory
tem.mem_ctrls.num_writes::cpu.itb.walker 2           # Number of write requests responded to by this memory
tem.mem_ctrls.num_writes::cpu.data 10467626         # Number of write requests responded to by this memory
tem.mem_ctrls.num_writes::pc.south_bridge.ide 46720   # Number of write requests responded to by this memory
tem.mem_ctrls.num_writes::total 10514348            # Number of write requests responded to by this memory
tem.mem_ctrls.bw_read::cpu.dtb.walker 38332         # Total read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_read::cpu.itb.walker 20478         # Total read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_read::cpu.inst 405191828           # Total read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_read::cpu.data 17480936            # Total read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_read::pc.south_bridge.ide 7272      # Total read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_read::total 422738847             # Total read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_inst_read::cpu.inst 405191828      # Instruction read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_inst_read::total 405191828        # Instruction read bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_write::cpu.itb.walker 3            # Write bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_write::cpu.data 14094010           # Write bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_write::pc.south_bridge.ide 583516   # Write bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_write::total 14677529             # Write bandwidth from this memory (bytes/s)
tem.mem_ctrls.bw_total::cpu.dtb.walker 38332        # Total bandwidth to/from this memory (bytes/s)
tem.mem_ctrls.bw_total::cpu.itb.walker 20481        # Total bandwidth to/from this memory (bytes/s)
tem.mem_ctrls.bw_total::cpu.inst 405191828         # Total bandwidth to/from this memory (bytes/s)
```

Co-Design Opportunities for OpenMP Tools (OMPT)

- It is impractical to simulate an entire program; we must pick *regions of interest* (ROI)
 - Slow downs for detailed tools like Gem5 are highly variable, 10,000x to 190,000x slow down
 - Less detailed tools such as ARMIE are much faster, but have less predictive capabilities
- OMPT can provide tool specific extensions to allow better integration between simulators and applications
 - Improves compatibility between different OpenMP implementations
 - Tools also allow a greater degree of control without having to use modified libomp.so or manually annotating code

Initial deployment of gem5

Simulation couples system simulation scripts with code regions of interest (ROI)

```
#Start code with AtomicSimpleCPU
./build/X86/gem5.opt configs/example/fs.py -b ValMemLat --disk /dist/m5/system/disks/linux-x86.img
...
#Execute in functional mode to first checkpoint
...
./build/X86/gem5.opt configs/example/fs.py -b ValMemLat --disk /dist/m5/system/disks/linux-x86.img
--checkpoint-restore=1 --restore-with-cpu=DerivO3CPU
~
~
```

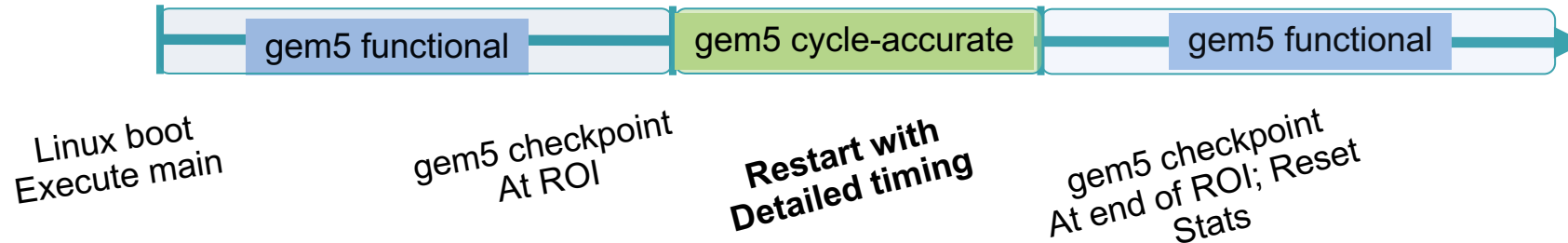
Gem5 Python Scripts

```
!Jacobi code segment
! ...
! Create an M5 checkpoint before the region of interest
m5_checkpoint(0,0)
!$omp parallel do private(xx,yy)
do j = 1,m
do i = 1,n
xx = -1.0 + dx * dble(i-1)      ! -1 < x < 1
yy = -1.0 + dy * dble(j-1)      ! -1 < y < 1
u(i,j) = 0.0
f(i,j) = -alpha *(1.0-xx*xx)*(1.0-yy*yy) - 2.0*(1.0-xx*xx)-2.0*(1.0-yy*yy)
enddo
enddo
!$omp end parallel do
! And another checkpoint after the region ends
m5_checkpoint(0,0)
```

Annotated Region of Interest

We use checkpoints to simulate ROIs and to mitigate up the large slow down versus native execution (~10,000x)

Program execution



Challenges for Selecting ROIs for Co-Design

- Difficult to programmatically pick regions of interest.
 - Many current techniques are intrusive (i.e., manual macro insertion) and are not conducive to reproducibility
 - Examples:
 - gem5 manual ROI and checkpointing, ArmIE ROI for memory tracing
 - Using different binaries for different simulators

```
[bzf@wombat-login2 gem5-tools]$ cat checkpoint.c
#include <gem5ops.h>

int main(int argc, char** argv)
{
    m5_checkpoint(0,0);
    return 0;
}
[bzf@wombat-login2 gem5-tools]$ armie -msve-vector-bits=2048 --iclient libinscount_emulated.so ./check
point
Client inscount is running
261312 instructions executed of which 0 were emulated instructions

Guest process terminated by signal: 15687 Illegal instruction ./checkpoint
[bzf@wombat-login2 gem5-tools]$
```


Problem Code

- Where to put check points?

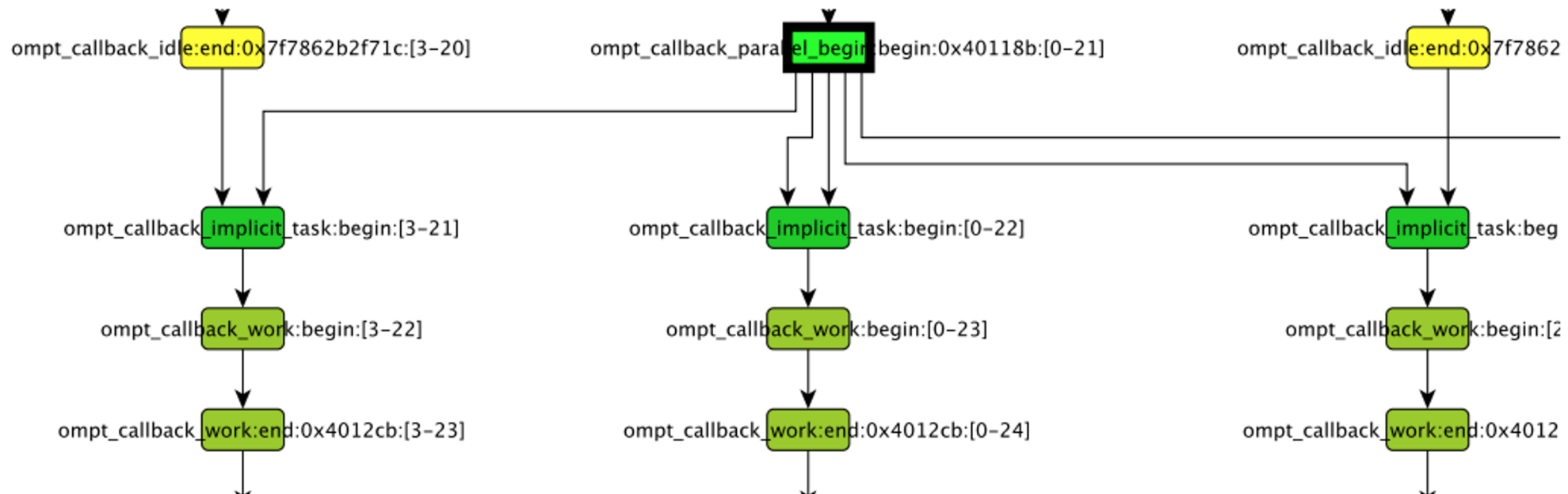
```
1  int i, j;
2
3  #pragma omp parallel private(j)
4  {
5      for (j=0; j < innerreps; j++)
6      {
7          /* Tell simulator that this is the start of a checkpoint */
8          m5_checkpoint(0,0);
9          #pragma omp barrier
10         #pragma omp for simd simdlen(8)
11         for (i=0; i < itersperthr * nthreads; i++)
12         {
13             C[i] = A[i] * B[i];
14         }
15         /* Second call tells simulator checkpoint is done */
16         m5_checkpoint(0,0);
17     }
```

How do we make codesign easier and more scalable?

- Use OpenMP directives to identify important regions of code that we want to simulate
 - Users have already identified them as important regions for performance
- Use tooling to identify and simulate regions of interest
 - OpenMP Tools (OMPT) interface to instrument applications at runtime
 - Handle incompatibilities between tools like gem5 and ARMIE
 - ARMIE is unhappy when it runs into gem5 magic instructions!
- Use OMPT based call backs to insert simulator specific magic insts
 - Allows for greater integration with simulators and runtime
 - Drop checkpoints after the barrier has synced instead of just before entering the barrier

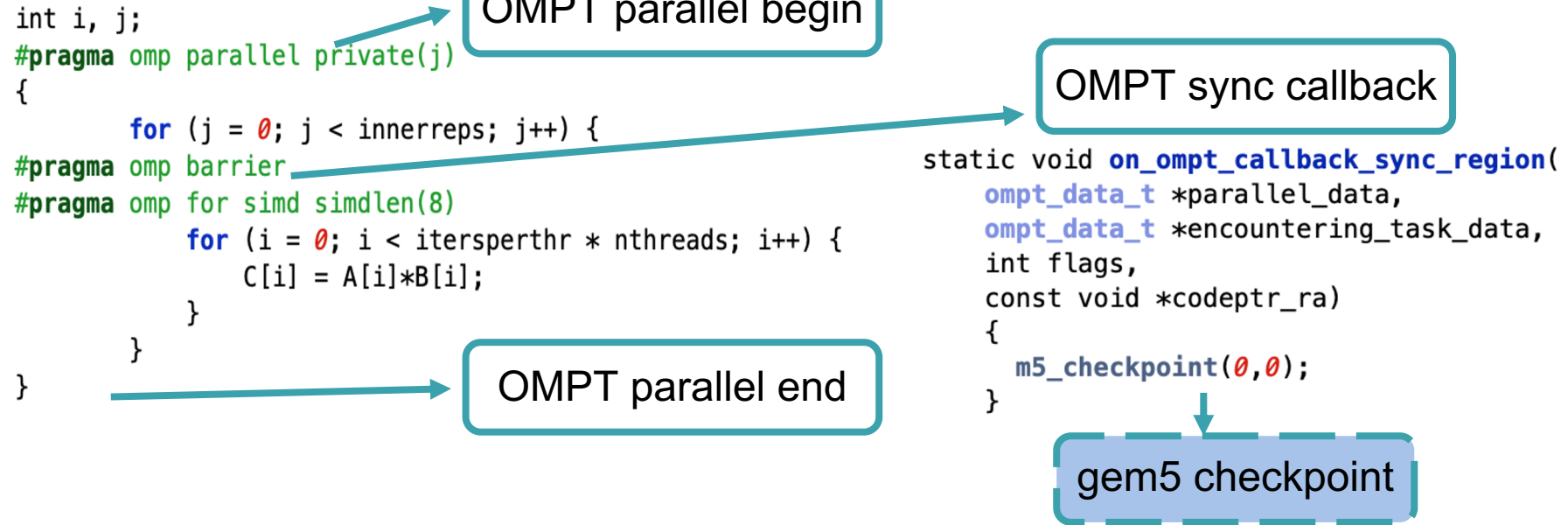
OMPT for runtime simulator hooks

- OMPT is a new tool interface for OpenMP 5.0+
- Allows tool developers to add analysis and introspection to the OpenMP runtime
 - No need to manually instrument code with gem5 ops or ARMIE ROI pragmas; OMPT callbacks can be used to add annotations



Example output from an OMPT visualization tool by Yonghong Yan, Philip Conrad, Yudong Sun, "Visualizing OpenMP Execution using OMPT". SC 2018

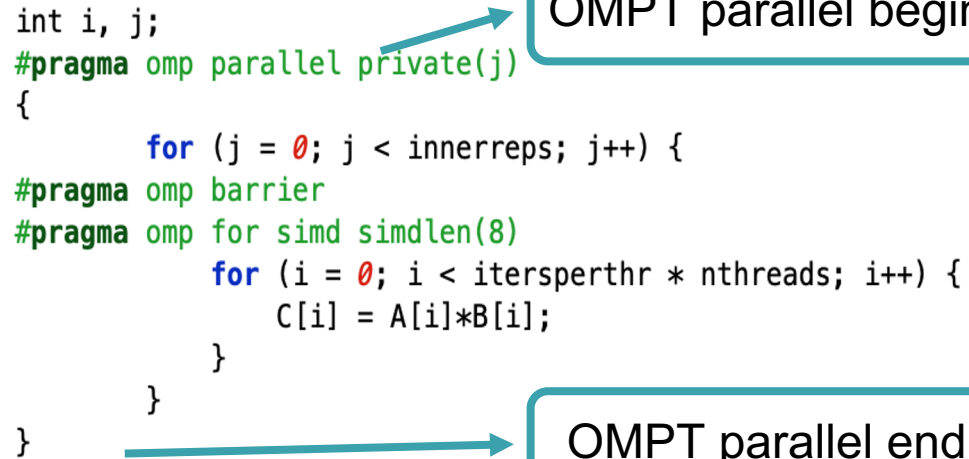
Extending OMPT for gem5 Simulations



- Initial work has used the OpenMP Tools API to drop gem5 checkpoints with simulators like gem5
- OMPT callbacks can be currently used to track *omp_parallel*, *omp_barrier*, *omp_single*

Extending OMPT for gem5 Simulations (2)

```
int i, j;
#pragma omp parallel private(j)
{
    for (j = 0; j < innerreps; j++) {
        #pragma omp barrier
        #pragma omp for simd simdlen(8)
        for (i = 0; i < itersperthr * nthreads; i++) {
            C[i] = A[i]*B[i];
        }
    }
}
```



OMPT parallel begin

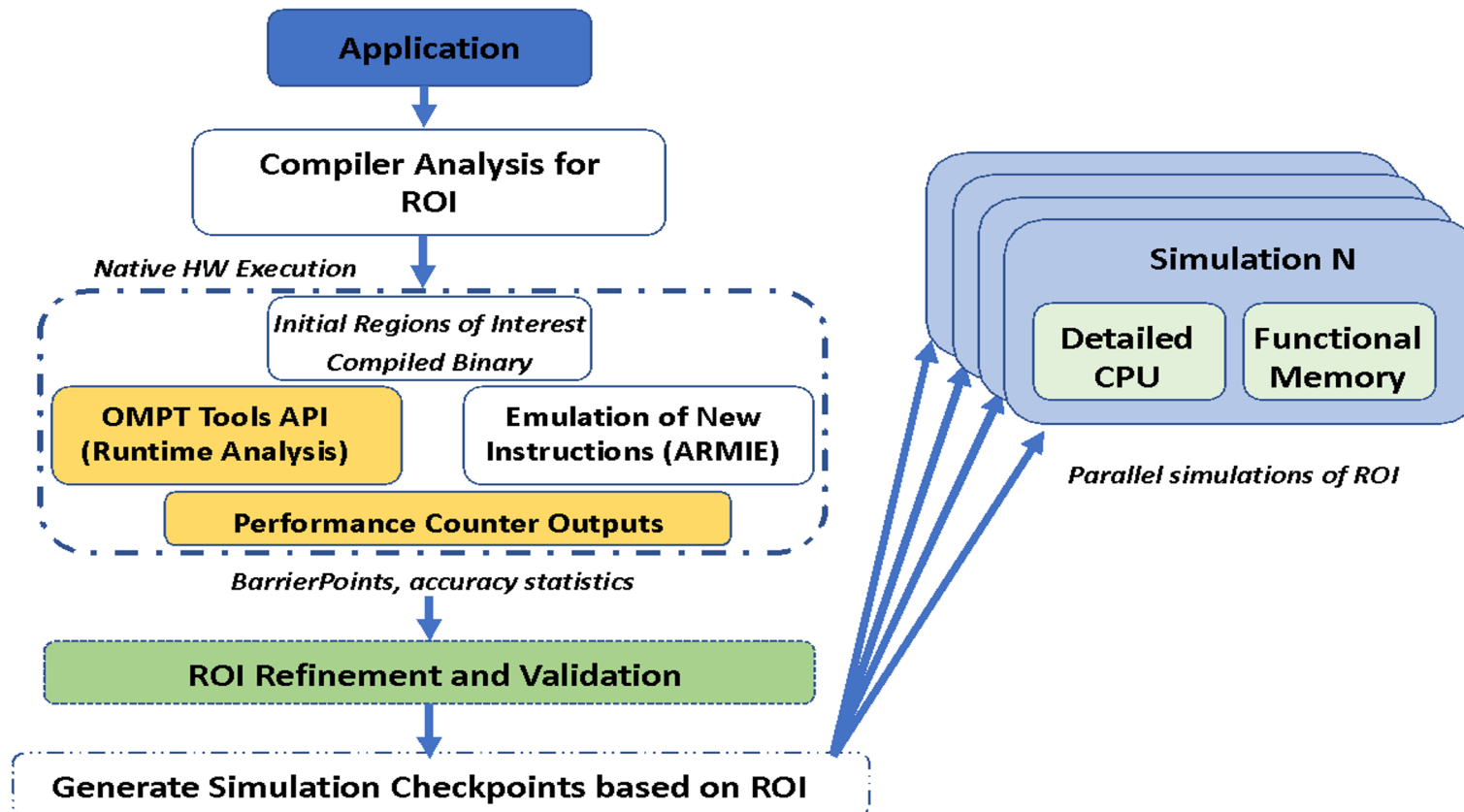
OMPT parallel end

```
static void
on_ompt_callback_parallel_begin(
    ompt_data_t *encountering_task,
    const ompt_frame_t *encountering_task_frame,
    ompt_data_t *parallel_data,
    unsigned int requested_parallelism,
    int flags,
    const_void *codeptr_ra)
{
    m5_switch_cpu();
}
```

- OMPT tools can be swapped out to provide different functionality!
 - Could switch between dropping checkpoints for ROI and resuming simulation with different CPU parameters

Architecture and Diagram for Simulation environment

- Integrate OMPT techniques with BarrierPoint work
- Develop additional OMPT based tools for improving simulation execution and instrumentation



Based on work by Miguel Tairum Cruz, Sascha Bischoff, Roxana Rusitoru,
“Shifting the Barrier: Extending the Boundaries of the BarrierPoint Methodology”. ISPASS 2018

Experiments

- We evaluate benchmark snippets from the EPCC microbenchmark suite
 - Overhead - performs tasks like OMP PARALLEL, OMP BARRIER, OMP FOR without any delay or computation in the parallel region
 - Syncbench - same as overhead but with added delay statements
 - SIMDBench - executes a basic parallel multiply using OMP simd pragmas
- All tests are run on an ARM ThunderX2 system where $N = \text{OMP_NUM_THREADS}$
 - Arm HPC Compiler 20 is used for compilation and ArmIE 20.0 is used for emulation tests
 - gem5-20 release is used for gem5 runs with a standard aarch64 CPU (simple memory model)

Code Snippets

SyncBench

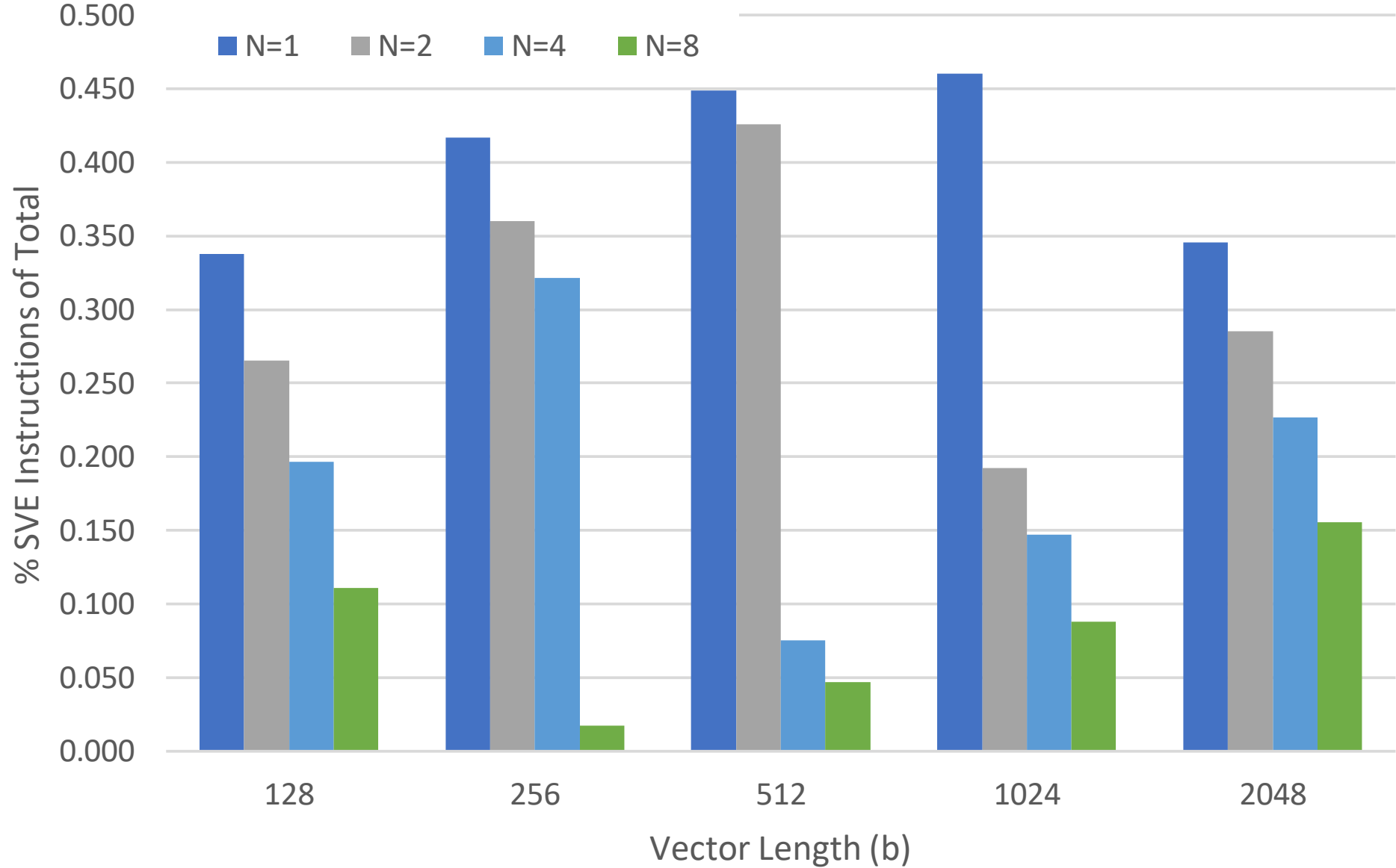
```
void testfor() {  
    __START_TRACE();  
    int i, j;  
    #pragma omp parallel private(j)  
    {  
        for (j = 0; j < innerreps; j++) {  
            #pragma omp for  
            for (i = 0; i < nthreads; i++) {  
                delay(delaylength);  
            }  
        }  
    }  
    __STOP_TRACE();  
}
```

*Note the manual trace option
for ArmIE - a good opportunity
for using OMPT!*

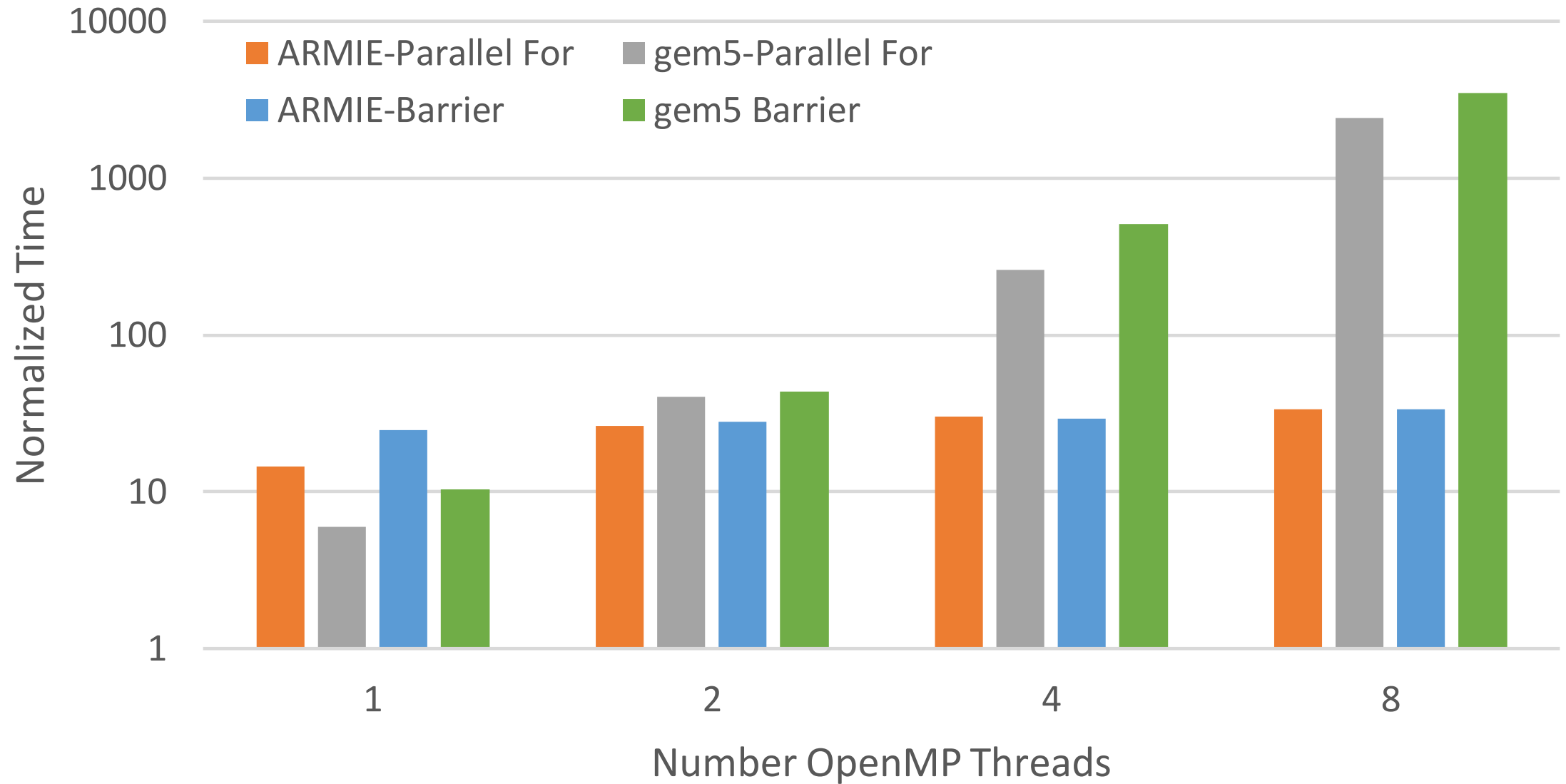
SIMDBench

```
void testsimd() {  
    int i, j;  
    #pragma omp parallel private(j)  
    {  
        for (j = 0; j < innerreps; j++) {  
            #pragma omp for simd simdlen(8)  
            for (i = 0; i < itersperthr * nthreads; i++)  
                C[i] = A[i]*B[i];  
        }  
    }  
}
```

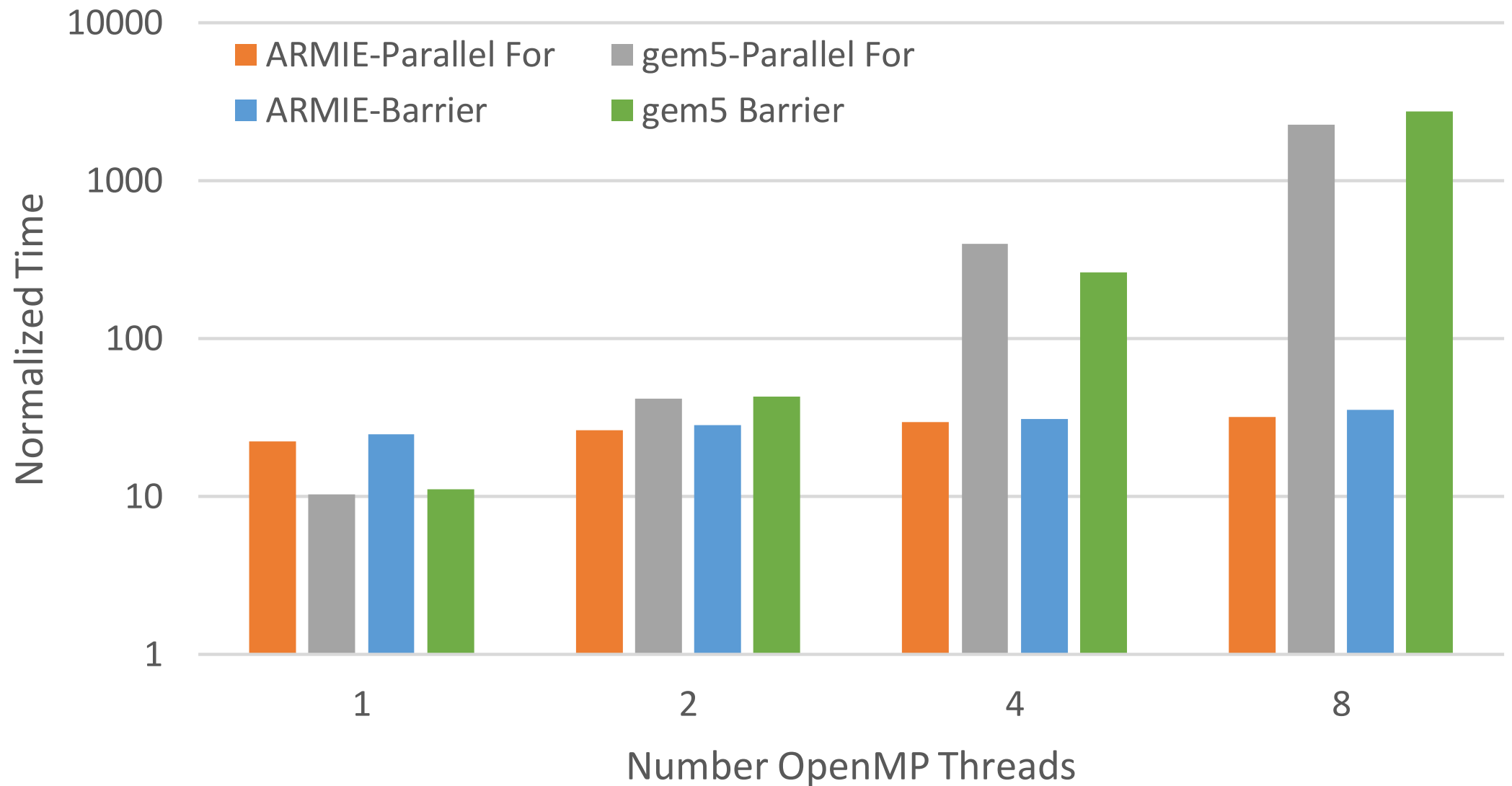
Measurements - SIMD Bench with ArmIE



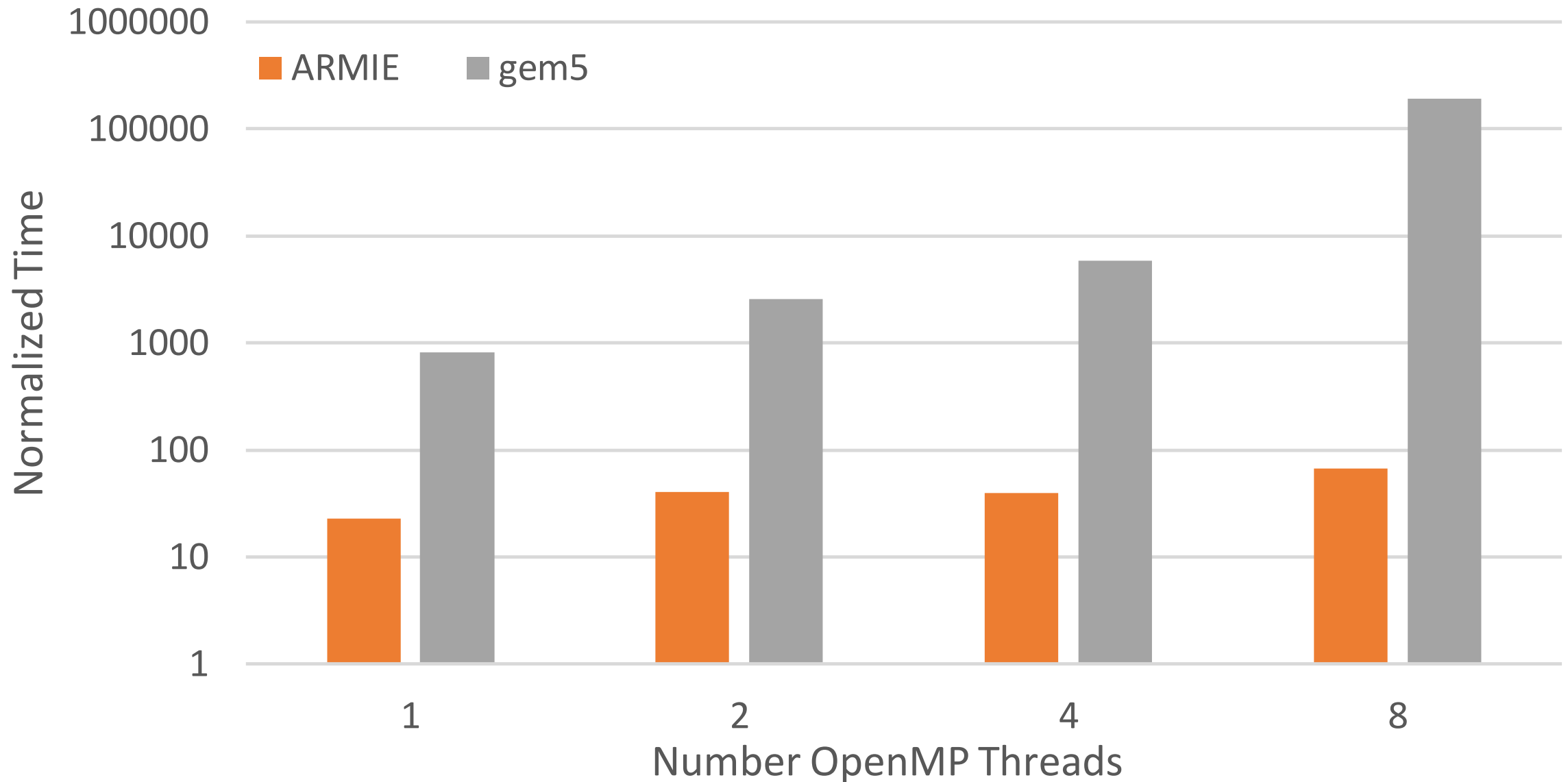
Overhead Measurements - ArmiE versus gem5



SyncBench Measurements - ArmIE versus gem5



Measurements - SIMDBench - ArmIE versus gem5 Runtime



Conclusions

- Architectural simulations are important for future codesign but are limited by execution speed
 - We can use OpenMP directives to identify ROI in applications to reduce codesign time
 - Once checkpoints are collected we can run simulations in parallel for ROI (*future work*)
- OMPT can be a valuable tool for seamlessly switching between emulation and simulation for ROI
 - Our work demonstrates how to integrate ArmIE and gem5 for collecting checkpoints and swapping architectural models for simulation
 - Sampling of OMPT callbacks can help mitigate performance overheads of gem5
- Future work will look at further integrating OMPT with gem5 and ROI tools like BarrierPoints (automatically map ROIs using OpenMP barrier regions)

What is a Barrier Point?

```
#pragma omp parallel for
for (j = 0; j < innerreps; j++)
{
    #pragma omp barrier
    for (i = 0; i < itersperthr * nthreads; i++)
    {
        C[i] = A[i]*B[i];
    }
}
```

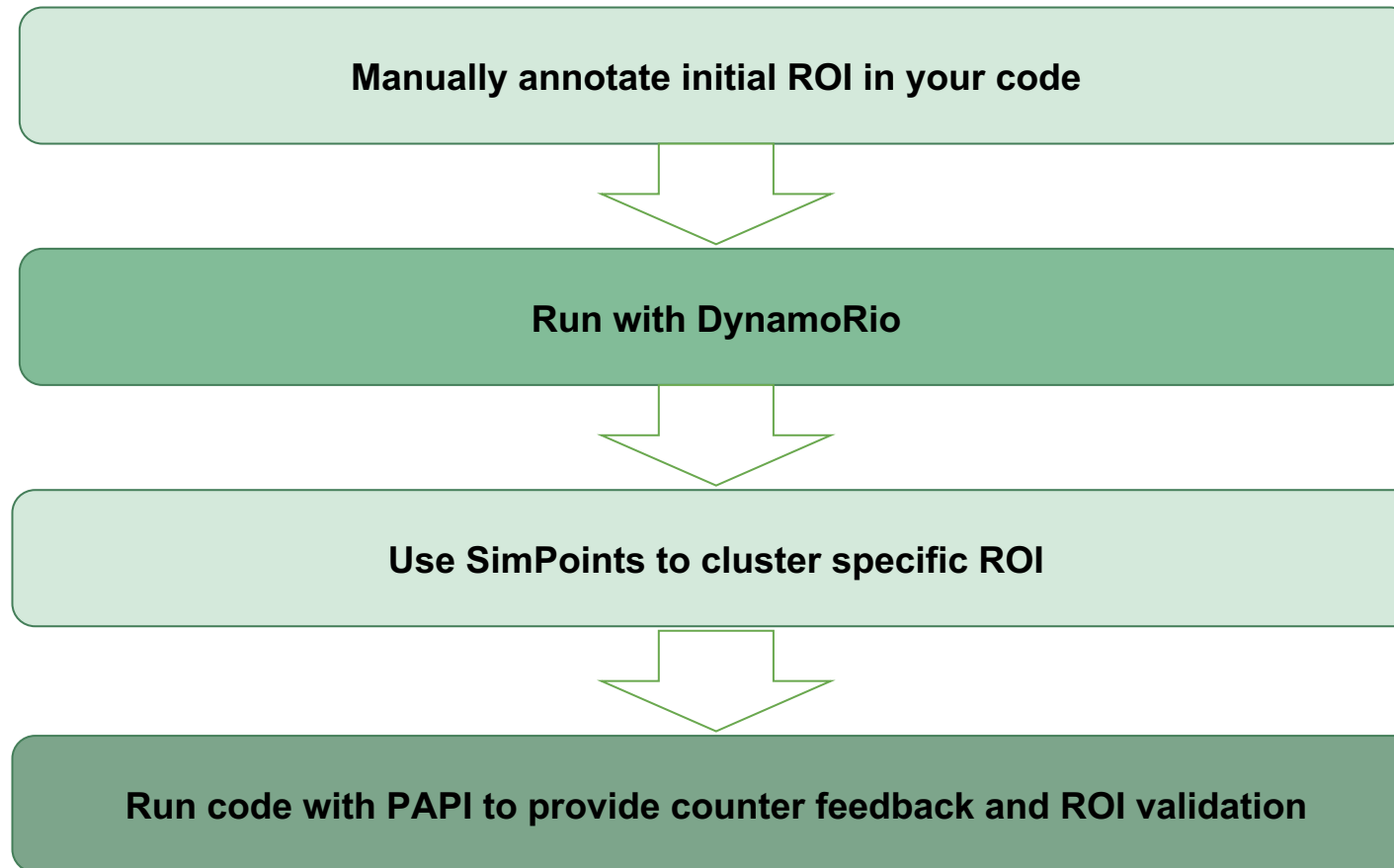
Implicit Synchronization Point

Explicit Synchronization Point

Implicit Synchronization Point

Based on work by Carlson, Trevor E., Wim Heirman, Kenzo Van Craeynest, and Lieven Eeckhout.
"Barrierpoint: Sampled simulation of multi-threaded applications." ISPASS 2014

Barrier Points Workflow

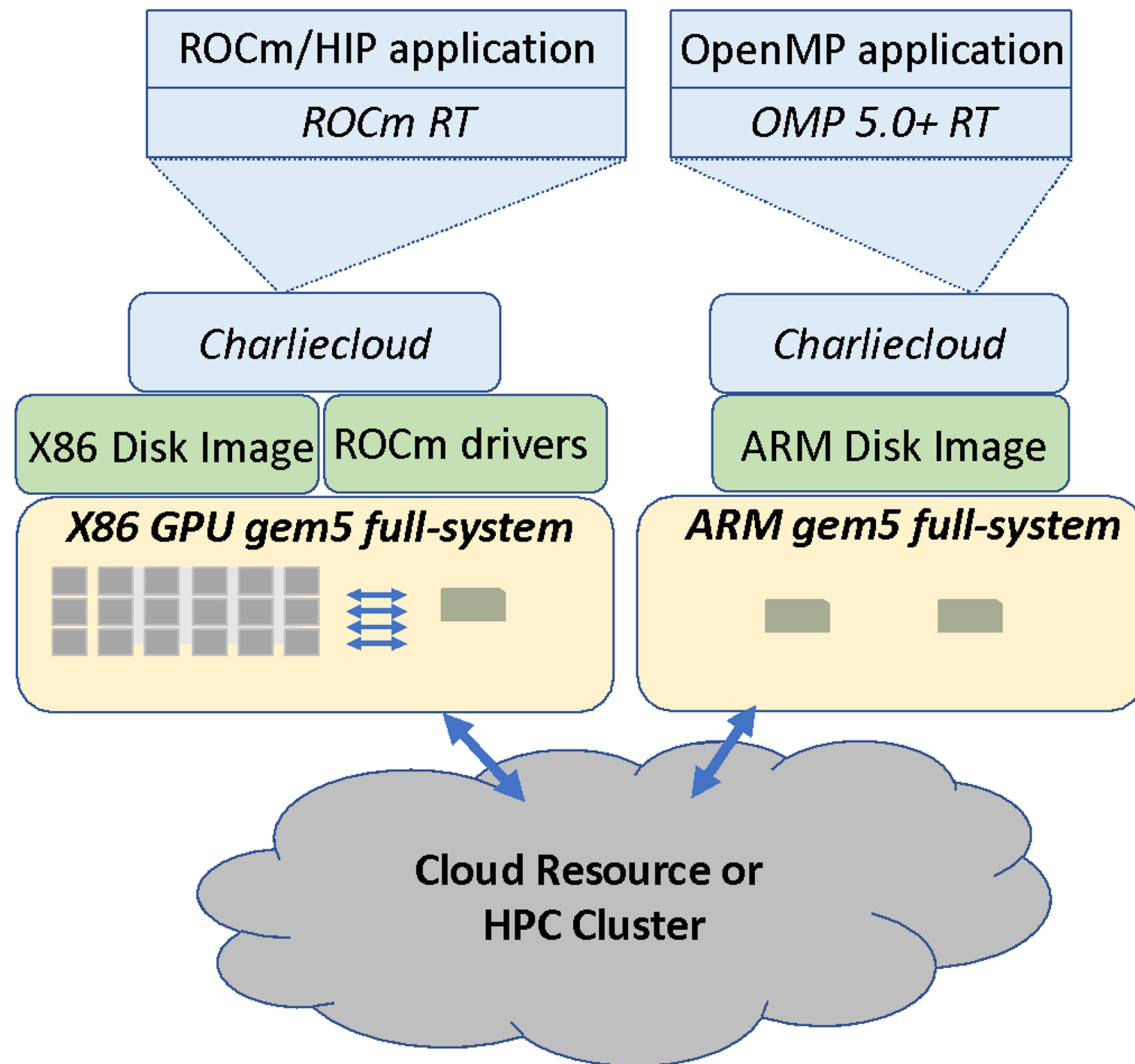


Why is the concept of BarrierPoints relevant for architectural simulation and OMPT?

- Gem5 has to be run with parallel configurations in an embarrassingly parallel fashion
- OMPT helps! it's a good way to extend and implement new BarrierPoint-like features
 - We envision that OMPT could be used to automatically add regions of interest as well as trigger some of the analysis BarrierPoints uses (e.g., trigger DynamoRio emulation or PAPI counters)

Future work: Deployment of Design Space Exploration at Scale

- Allow users to create their own lightweight simulation containers inside FS simulations
- Mirrors current use-cases for systems like Summit (Docker, Singularity) and reduces deployment overheads



Creating a Consistent Environment for Simulations

- Ensuring consistent behavior of mini-apps between environments
- Simulators often have their own quirks that need to be managed
- Leverage containers to ensure a consistent runtime environment
- Simulated applications can be bundled as containers
 - Easier to distribute
 - Same binary between simulators
 - Reproducible

Future work: Charliecloud

- While most container environments have heavy requirements and root privileges, Charliecloud does not
 - Simple to include in gem5 full system simulation or run in HPC centers
 - Just need its ch-run binary!
 - Container image is flattened into tarball
 - Extract into new directory and container environment is a ch-run away
 - Host environment tools can be leveraged with bind mounts
 - ARMIE home dir can be bind mounted into container directory
 - ARMIE tools are then available inside isolated, user defined environment
 - gem5 can bind gem5 OMPT-enabled tool directory into simulations

```
[bzf@wombat-login2 slides]$ tar xf simdbench.tar.gz -C simdbench
[bzf@wombat-login2 slides]$ ch-run --bind $ARMIE_PATH simdbench -- /mnt/0/bin64/armie -msve-vector-bit
s=2048 --iclient libinscount_emulated.so /app/simdbench
Client inscount is running
SVE: 0x00000000000401688 0x043f57df
```

Example ch-run with ARMIE bind mount from host

Acknowledgments

- This work was funded by an Oak Ridge National Laboratory Directed Research & Development (LDRD) project
- This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.