

# Introduction to the Sitar Framework

for Parallel Simulation of Synchronous Performance Models

**Dr. Neha Karanjkar**

Assistant Professor, CSE

Indian Institute of Technology Goa (IIT Goa)

[nehak@iitgoa.ac.in](mailto:nehak@iitgoa.ac.in)

<https://nehakaranjkar.github.io>



# Context, Overview and Purpose of this talk

- Types of models I'm going to focus on
- Need for Parallel Simulation .... and the reason that it can be Non-Trivial
- Sitar, its Features and Performance Results

# The Need for Parallel Simulation

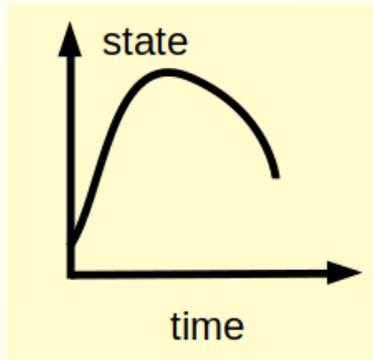
- **The Bad:** Models are getting larger, Benchmarks are getting longer
- **The Good:** Host systems are getting larger too! (many-cores, clusters, GPUs)
- **The Ugly:** There is something **fundamentally sequential** about dynamic simulation (in most cases). Parallelizing is non-trivial.

# The Need for Parallel Simulation

- **The Obvious:** Design exploration can still be sped up (run many simulations in parallel). Single simulation time can still bottleneck design iterations!
- **The Opportunity:** Large models can be partitioned in domain-space for some parallelization. ...is linear scaling possible?
- **Technical Challenge:** Maintaining causality, deterministic execution.

## Continuous System

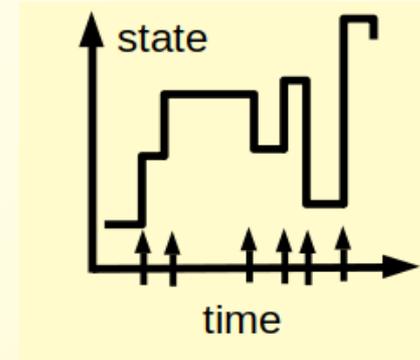
State of the system can evolve continuously with time



System behavior typically described as ODEs/PDEs

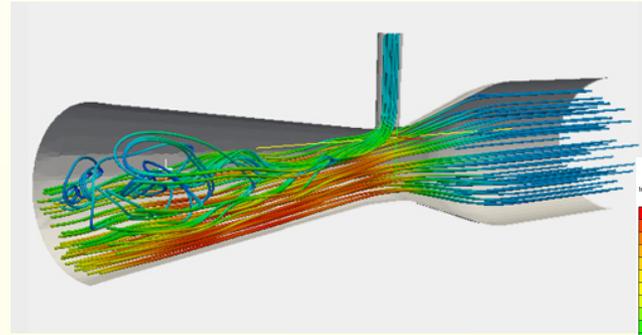
## Discrete-Event System

State changes assumed to happen only at discrete time-instants (called **Events**)

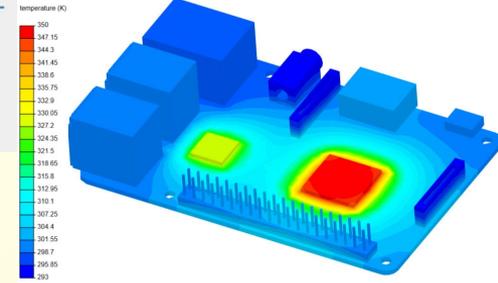


System behavior described as causality between events and time delays between them

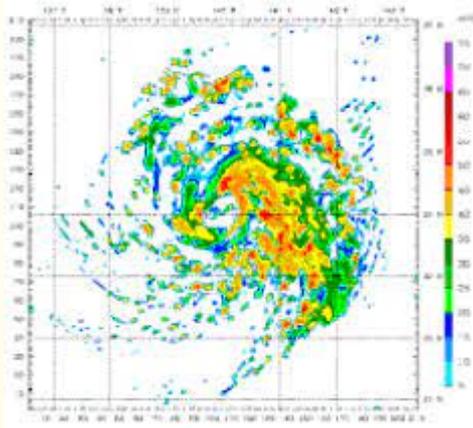
# Some Applications of Continuous System Simulations



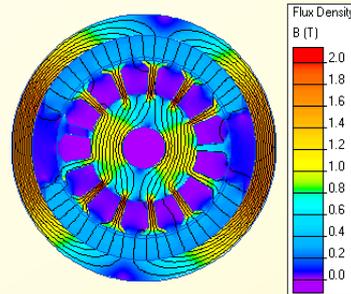
Flow simulations



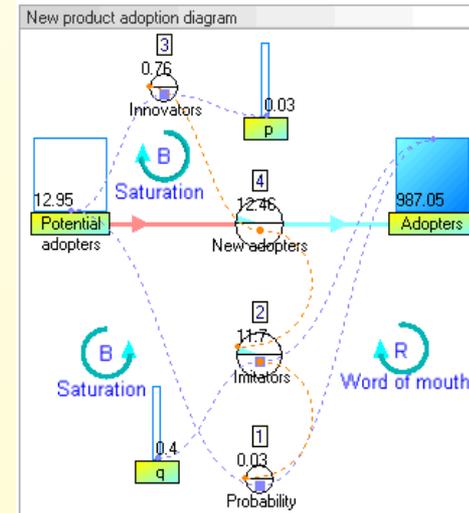
Weather simulation



Electro-mechanical simulation

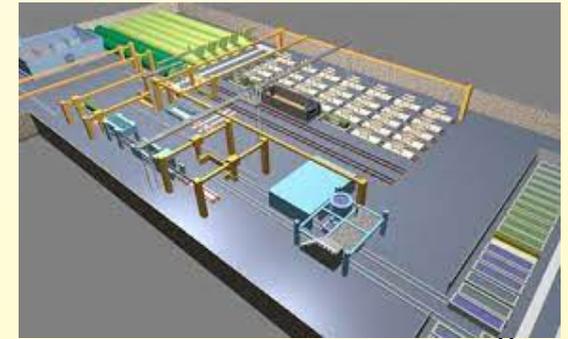
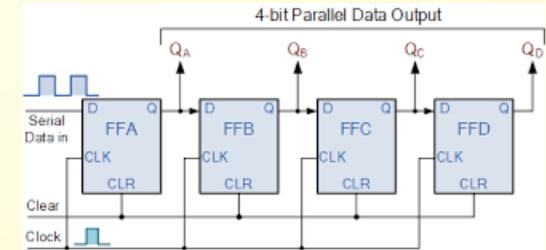
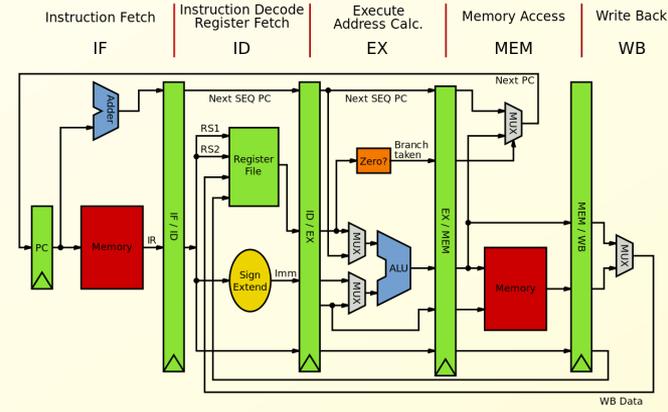
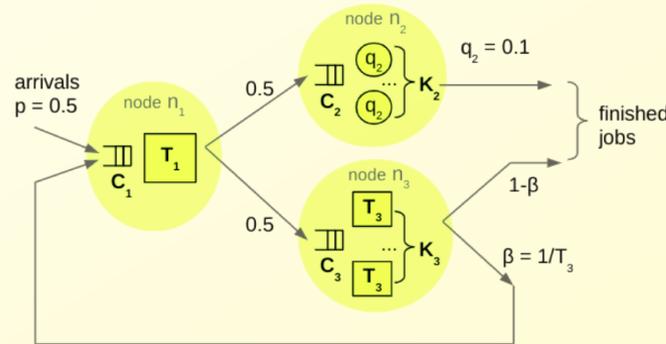
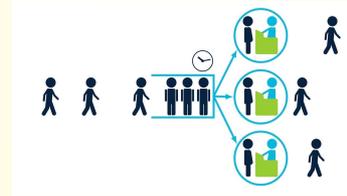
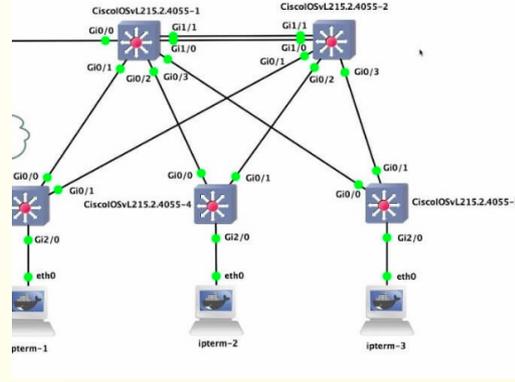


System Dynamics Simulation



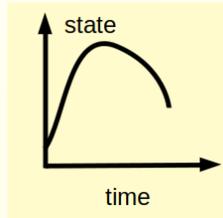
# Discrete-Event Simulation Applications

- Computer networks
- Digital systems
- Computer architecture
- Queueing networks
- Factory/Assembly lines
- Models of customer arrival, behavior, inventory, sales etc in retail, airports, restaurants ..



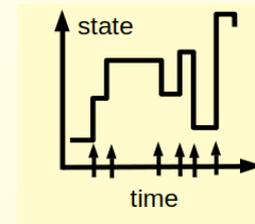
## Continuous System Model

**State** of the system is assumed to evolve **continuously** over time



## Discrete-Event System Model

**State** changes assumed to happen only at **discrete time-instants** (called **Events**)



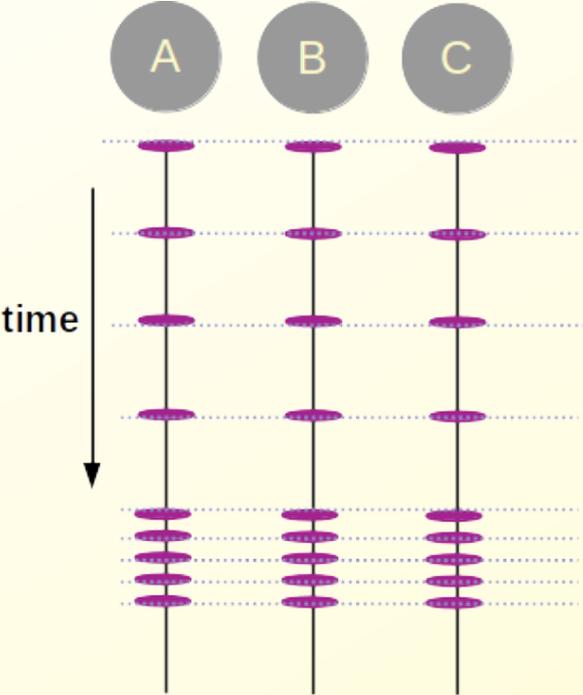
**The difference is conceptual. In practice...**

- Continuous simulators (on digital computers) **have to** discretize the state-space and time axis.
- The body of techniques and approaches, challenges, history, and domain experts are generally separate between these two areas.

# Continuous Simulation

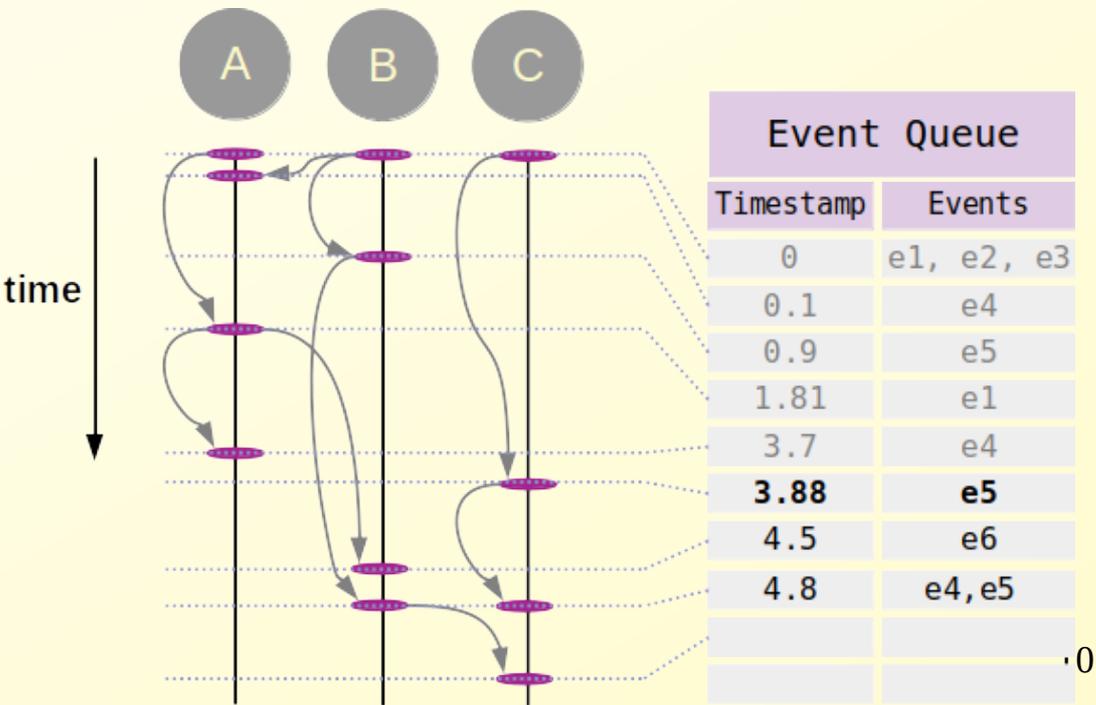
discretization

Time-stepped Approach



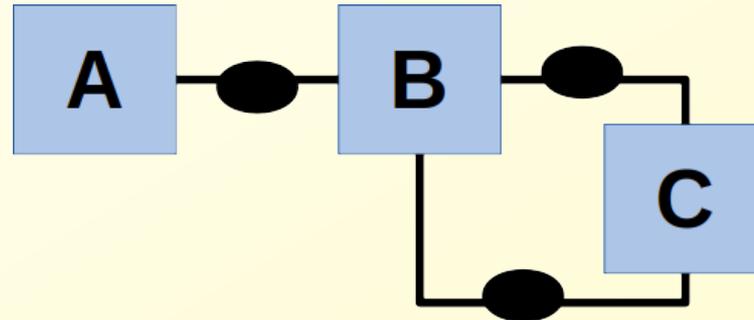
# Discrete-Event Simulation

Event-driven Approach



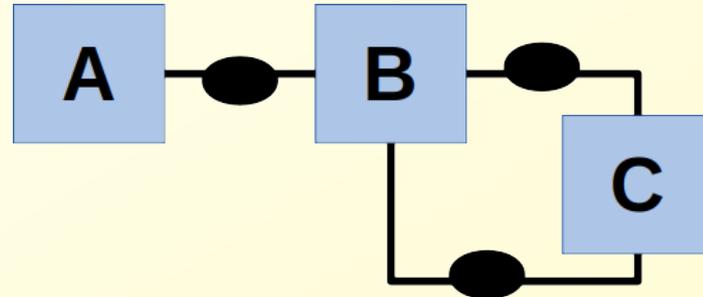
# Synchronous / Cycle-based Models

- All components operate on a single global “clock”. All event times are integer-valued
- System can consist of concurrent components with static or dynamic communication structures



# Cycle-based Simulation

- Execute the behavior of each component in every clock cycle?
- Order of execution among modules matters!
- Race conditions on updating communication channels?

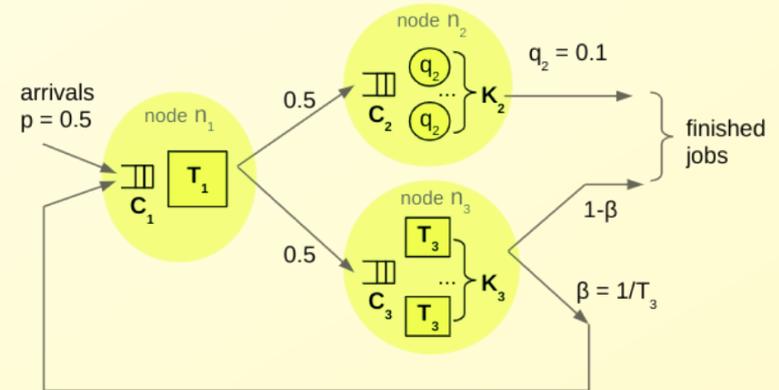
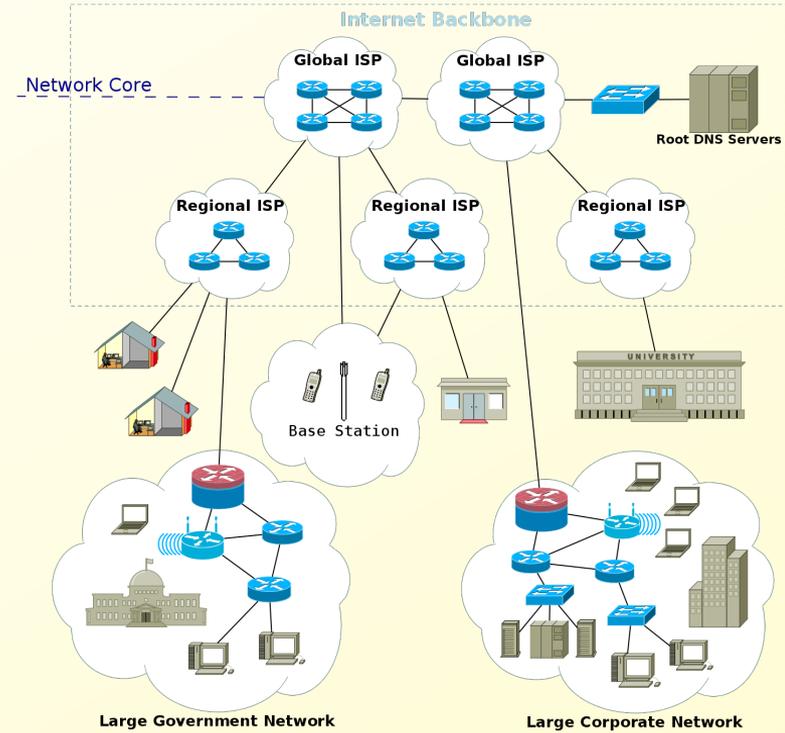
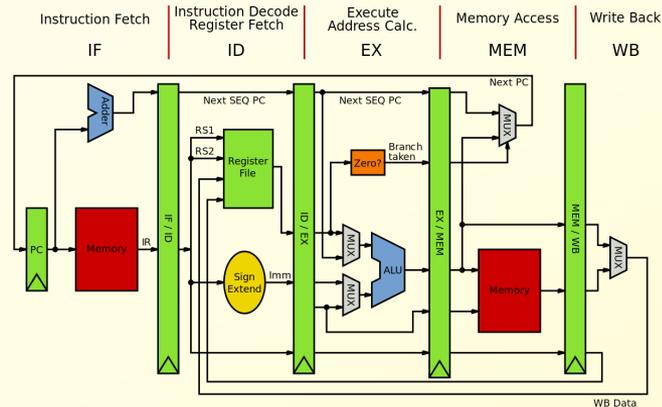
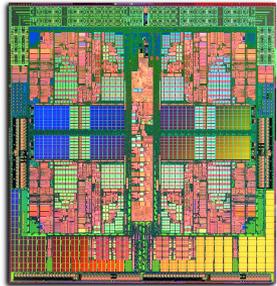


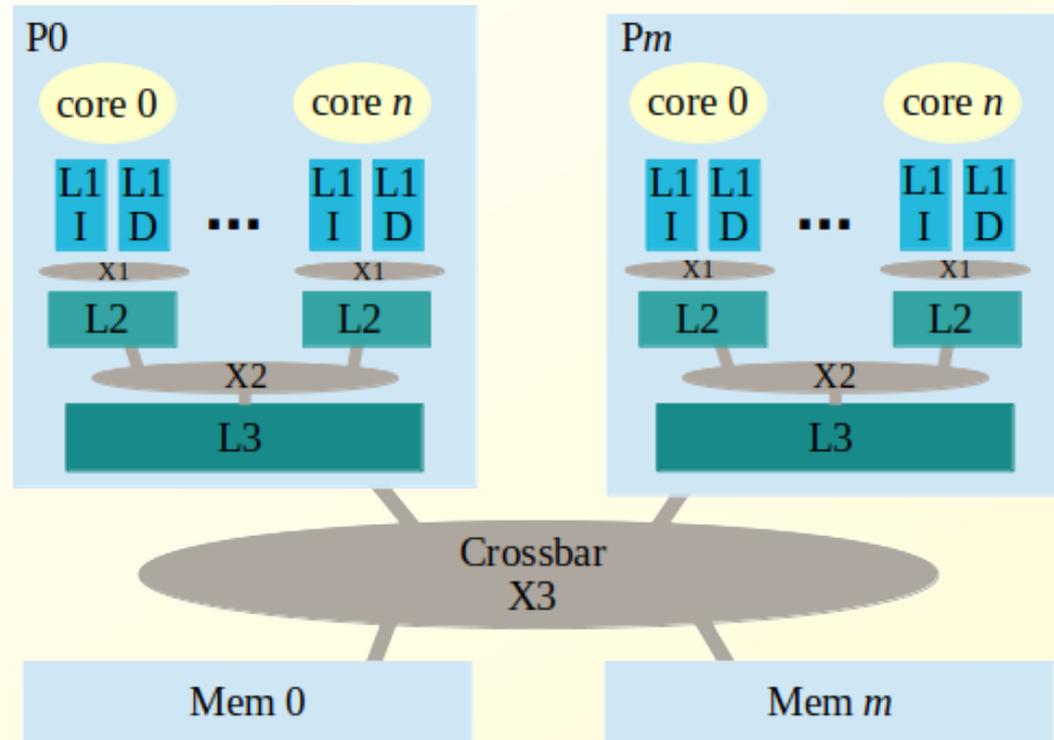
# Sitar: A modeling and parallel simulation framework targeted for systems ...

- that are large, with many concurrent components having complex, distinct behaviors,
- have a static structure (hierarchy and interconnections),
- operate in discrete-time, and
- where the communication between components incurs some delay...

# Some Examples:

- Computer Networks
- Architecture-level models for processors and Systems-on-Chip
- Clocked VLSI systems
- Discrete-time Queueing Networks
- ...





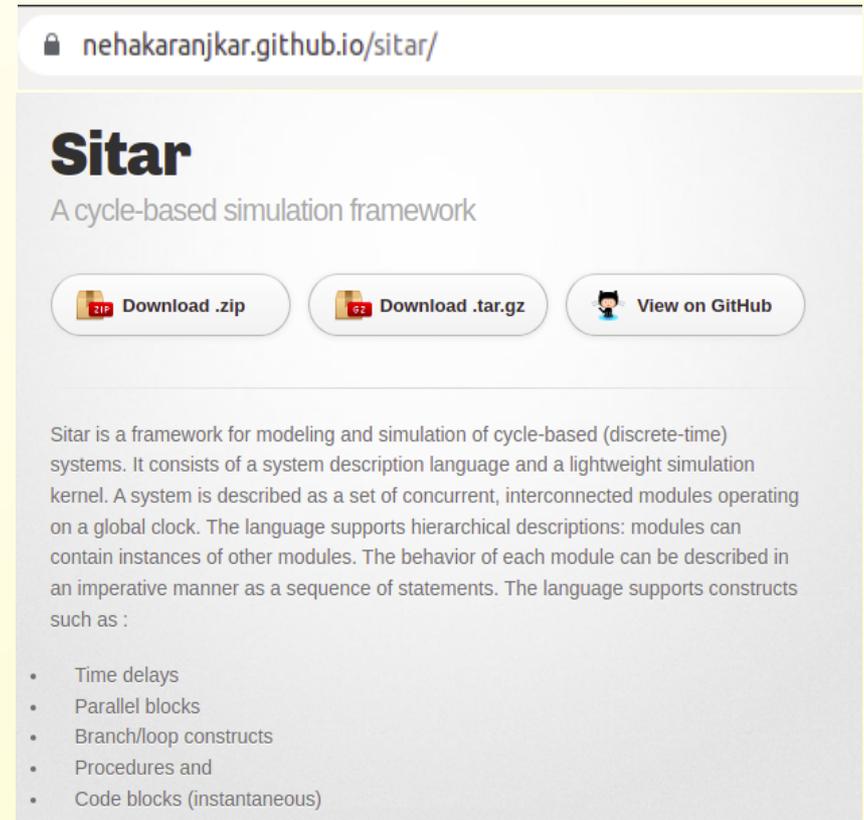
# About Sitar

- An open-source cycle-based modeling framework
- Targeted for writing quick architecture-level models for **fast, parallel simulation** on shared-memory systems
- Consists of **a Modeling Language** and **a Cycle-based Simulation kernel**

# Development

- Version 1 Developed as an internal tool for computer architecture research at IIT Bombay. (2013)
- Used in multi-core design exploration studies and simulating and optimizing queueing networks (2013-2015)
- Version 2: Open-source (MIT-License)

<https://sitar-sim.github.io/sitar/>

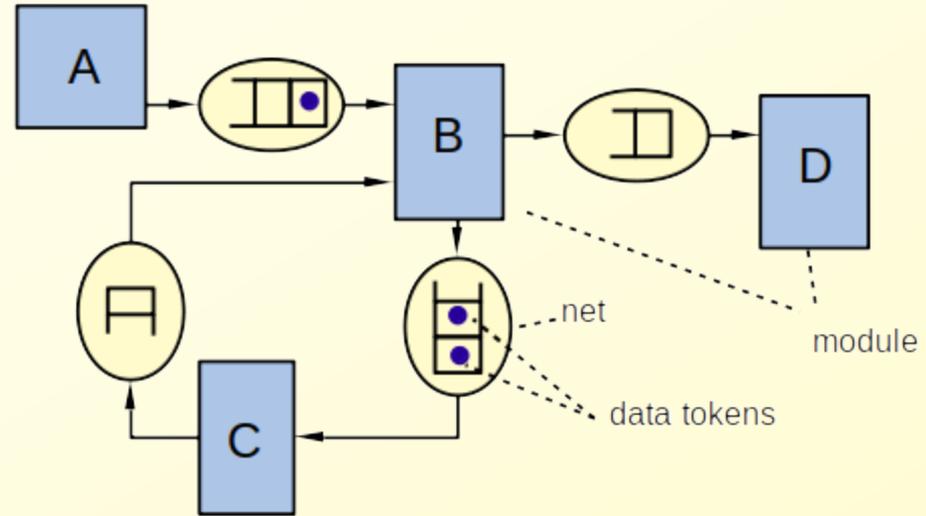


The screenshot shows the GitHub repository page for Sitar. The URL in the browser address bar is `nehakaranjkar.github.io/sitar/`. The repository name is **Sitar**, with the subtitle "A cycle-based simulation framework". Below the title are three buttons: "Download .zip", "Download .tar.gz", and "View on GitHub". The main text describes Sitar as a framework for modeling and simulation of cycle-based (discrete-time) systems, consisting of a system description language and a lightweight simulation kernel. It lists several constructs supported by the language:

- Time delays
- Parallel blocks
- Branch/loop constructs
- Procedures and
- Code blocks (instantaneous)

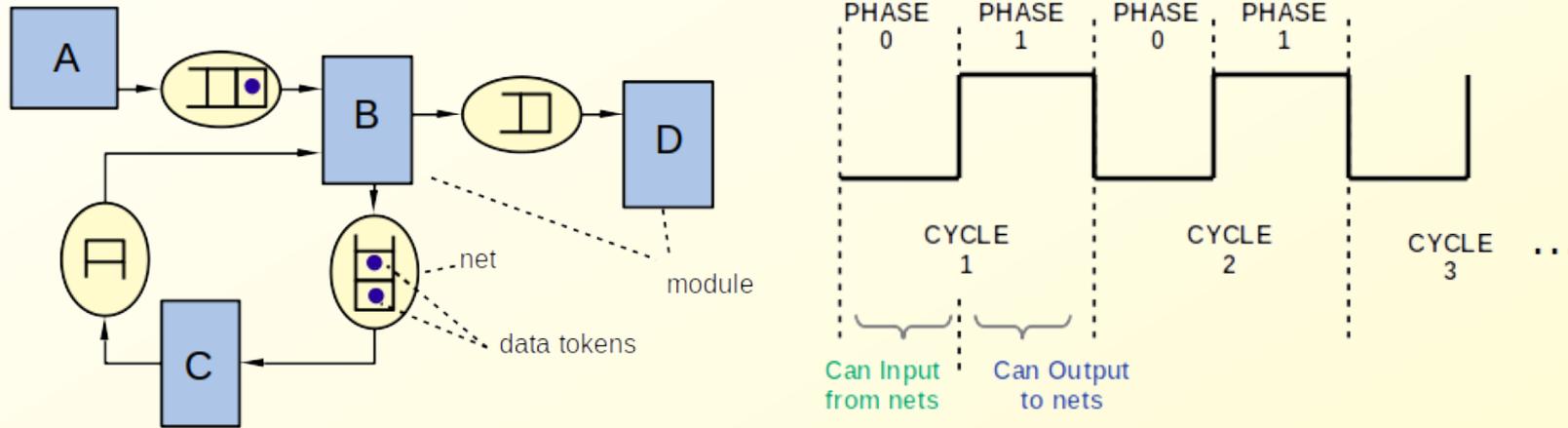
# Sitar Execution Model

- A system consists of *modules* communicating over FIFO channels (*nets*) by transfer of *data-tokens*
- All modules operate on a single clock



# Sitar Execution Model

- Two-phase execution (deterministic) : input in phase 0, output in phase 1



- Restriction:** Delay of at-least one cycle for information to propagate from the input of a module to it's output. (Communicating Moore machines paradigm)

# Sitar Execution Model

- Two-phase simulation algorithm (pseudo-code)

```
cycle = 0
while (cycle < simulation_end_time)
{
    phase=0;//run all modules in phase 0
    for (m=0; m<num_modules; m++)
        module[m].run(cycle, phase);

    phase=1;//run all modules in phase 1
    for (m=0; m<num_modules; m++)
        module[m].run(cycle, phase);

    cycle = cycle + 1
}
```

# Sitar Execution Model

- Two-phase simulation algorithm - Parallelized using OpenMP

```
cycle = 0
while (cycle < simulation_end_time)
{
    phase=0;
    #pragma omp for //run all modules for phase 0
    for (m=0; m<num_modules; m++)
        module[m].run(cycle, phase);
    #pragma omp barrier

    phase=1;
    #pragma omp for //run all modules for phase 1
    for (m=0; m<num_modules; m++)
        module[m].run(cycle, phase);
    #pragma omp barrier
    cycle = cycle + 1
}
```

# Sitar Modeling Language

- **Describing Structure:** Interconnection of modules and nets, Hierarchy, Generics (templated modules), For-loops to create regular structures
- **Describing Module Behavior:** In an imperative manner with conditional wait statements and delays, branches, loops, fork-join parallel blocks and embedded code
- **Other features:** Logging support, debugging support, syntax highlighting
- **Translation:** Each module description gets translated to a C++ class, compiled together with a lightweight simulation kernel to get an executable.

# Sitar Modeling Language: Structure

```
module Top
  //Instantiate a shift register
  //with 3 stages, and a per-stage delay of 1
  submodule S : ShiftRegister<3,1>

  //Instantiate the Producer and Consumer
  submodule P : Producer
  submodule C : Consumer

  //Connect the Producer and Consumer
  //to the two ends of the ShiftRegister
  P.out_port => S.n[0]
  C.in_port  <= S.n[3]
end module
```

```
module Stage
  parameter int DELAY = 1
  inport in_port
  output out_port
  behavior
    // describe the behavior of each stage
    //...
  end behavior
end module
```

```
module ShiftRegister
  //declare module parameters
  //and their default values
  parameter int N = 1 //number of stages
  parameter int DELAY = 1 //delay of each stage

  //The ShiftRegister consists of N stages
  submodule_array stage[N] : Stage<DELAY>
  //N+1 nets to connect all stages
  net_array n[N+1] : capacity 1

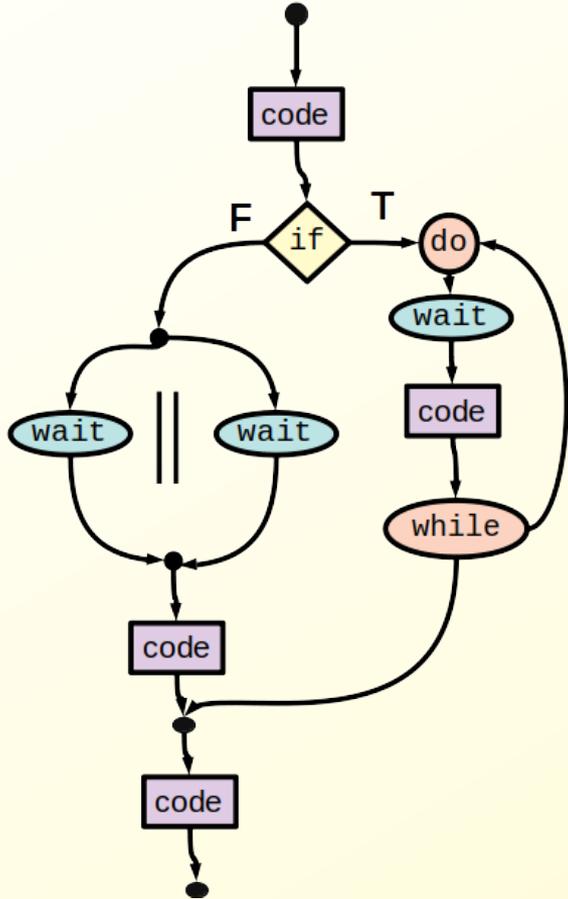
  //connect the stages via nets
  for i in 0 to (N - 1)
    stage[i].in_port  <= n[i]
    stage[i].out_port => n[i+1]
  end for
end module
```

# Sitar Modeling Language: Behavior

**Module Behavior:** described in an imperative manner as a sequence of statements

- **Atomic statements (instantaneous)**
  - wait (time), wait until (condition)
  - C++ code blocks
  - simulation control, logging
- **Compound statements (in-turn contain a sequence)**
  - do-while, if-else
  - Parallel blocks (fork-join concurrency)
  - Procedures

# Modeling Language: Behavior



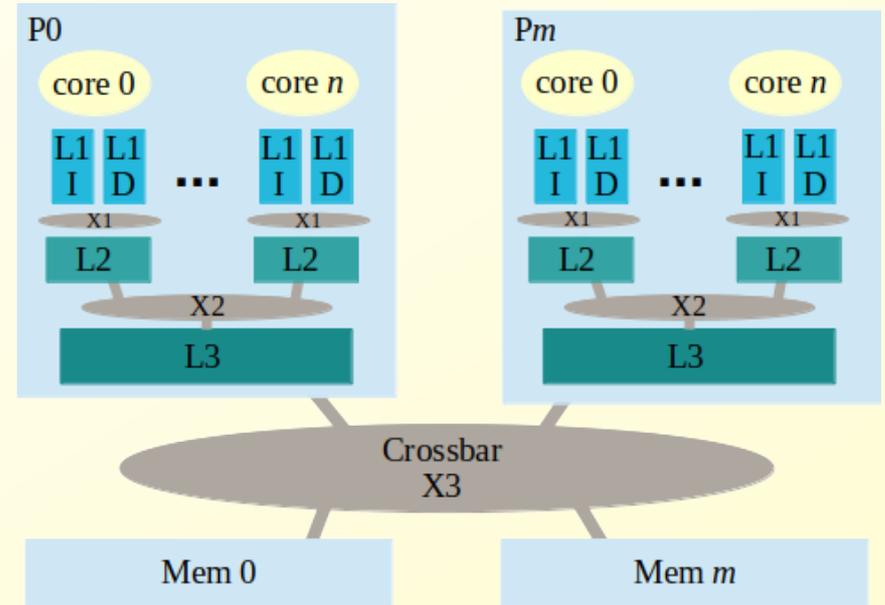
```
module Top
behavior

  decl $int x;$;
  //Input x from user
  $ std::cout<<"\nEnter a number:";std::cin>>x;$;

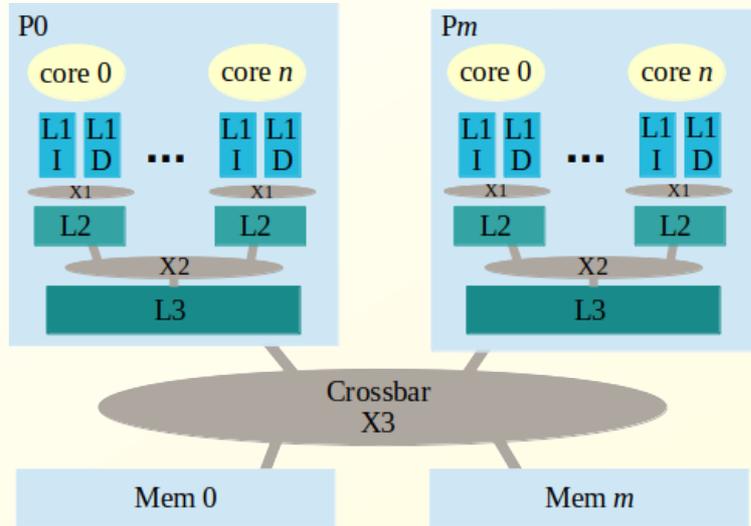
  //Branch
  if($x%2==0$) then
    do //Loop
      wait(1,0); //wait for one cycle
      $log<<endl<<"In the even branch";$;
      while($current_time.cycle()<4$) end do;
    stop simulation;
  else
    [ //Parallel block
      wait (2,0);
      ||
      wait until ($current_time>=time(x,0$));
    ];
    $log<<endl<<"In the odd branch";$;
  end if;
  $log<<endl<<"Done";$;
  stop simulation;
end behavior
end module
```

# Use Case and Scalability

- Parameterized Multi-core model with 3-level cache hierarchy
- Cores: in-order, Sparc V8 ISA
- Caches: per-core split L1, unified L2, shared L3
- Directory-based hierarchical MESI protocol
- Can run Sparc v8 binaries



# Use Case and Scalability

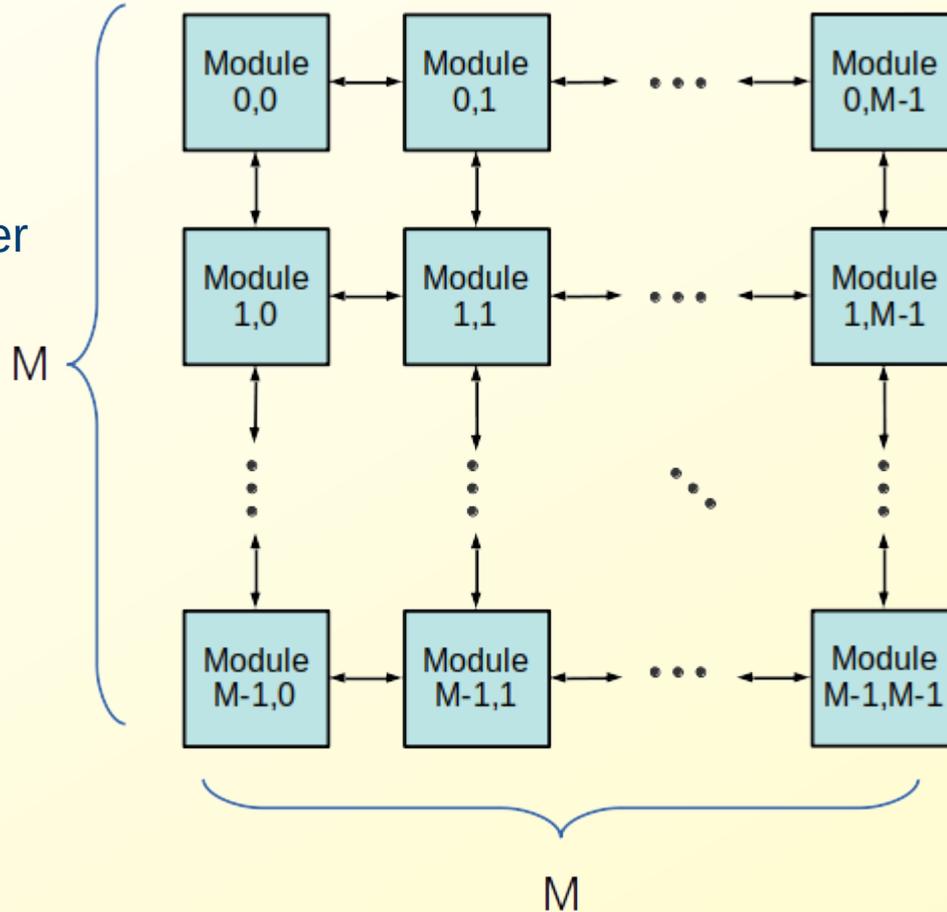


Kernel	Problem Size
Embarrassingly Parallel (EP)	$2^{16}$
Multigrid (MG)	$16^3$
3-D FFT PDE solver (FT)	$32^3$
Integer Sort (IS)	$2^{14} + 2^{12}$

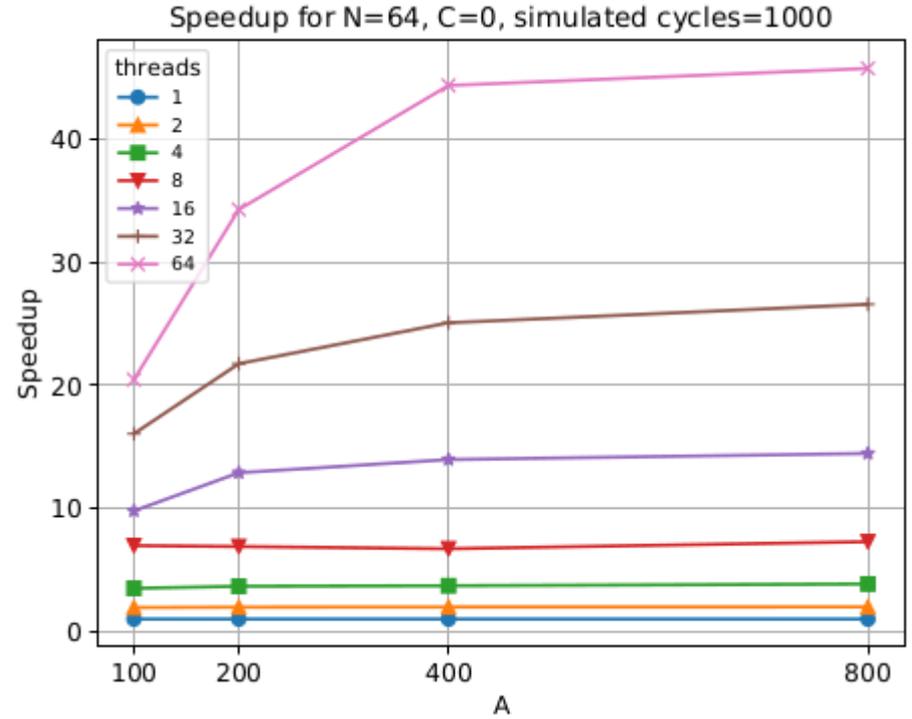
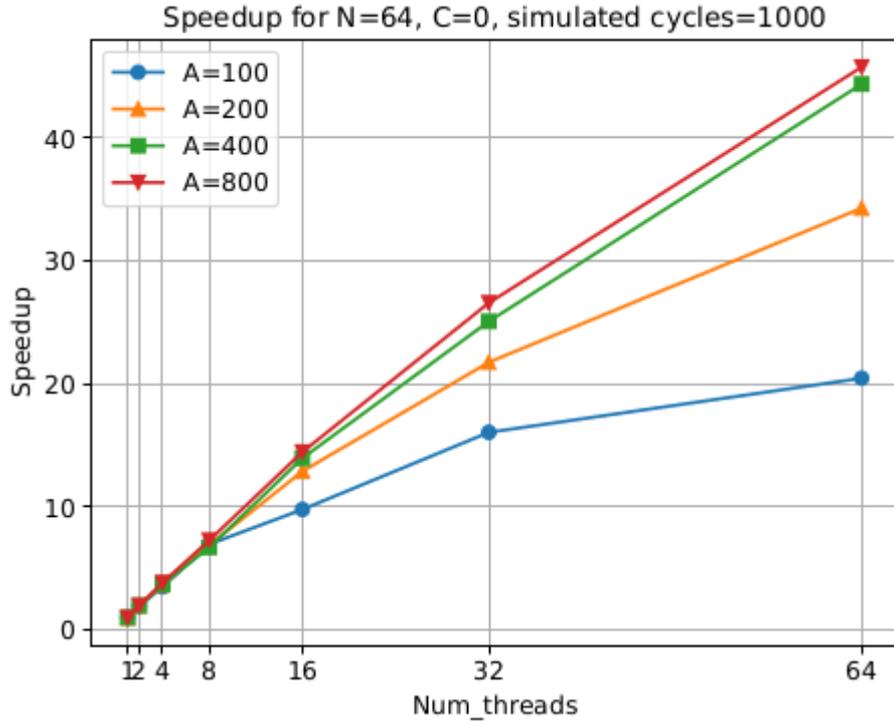
Workload (Sim Cycles)	Simulation time in seconds and (Speedup)		
	1 thread	2 threads	4 threads
EP (9781300)	954 (1x)	622 (1.5x)	442 (2.2x)
FT(16578568)	1710 (1x)	1112 (1.5x)	855 (2x)
IS(1434771)	156 (1x)	102 (1.5x)	78 (2x)
MG(13728356)	1452 (1x)	944 (1.5x)	726 (2x)

# Parameterized Benchmark Model

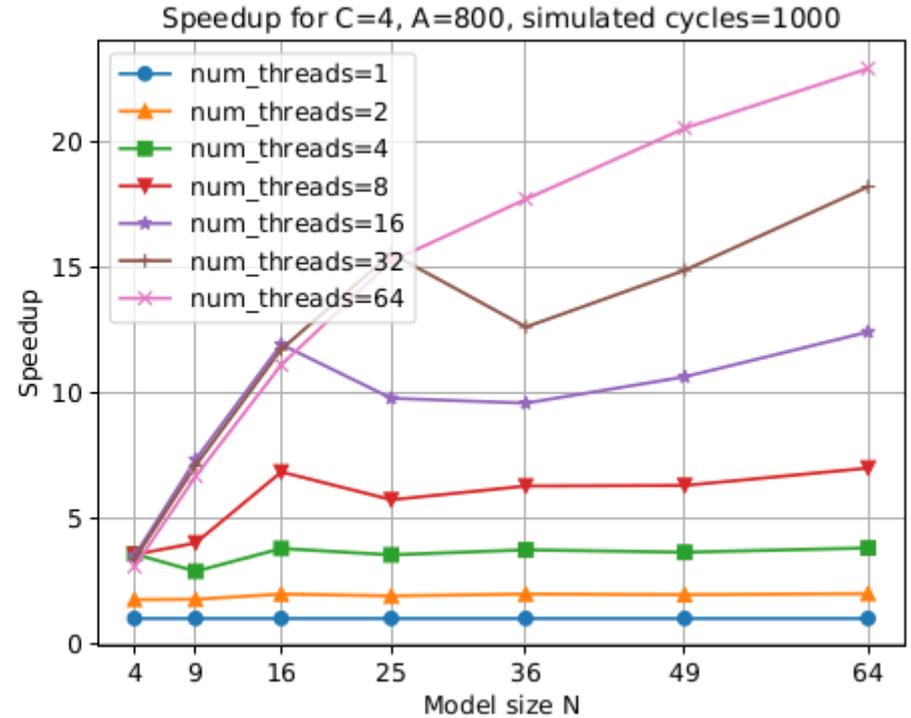
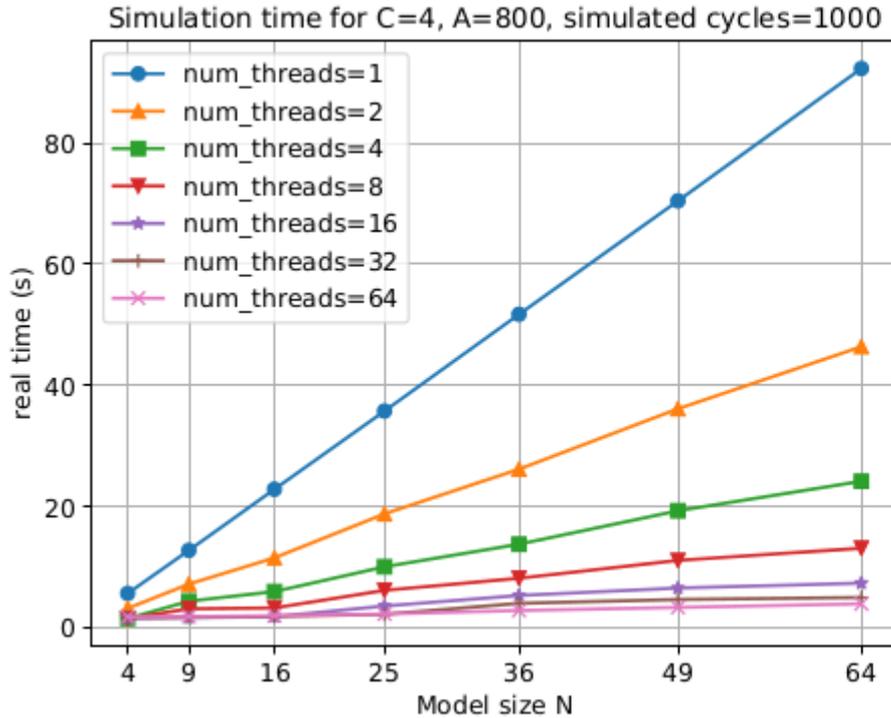
- **N** =  $M \times M$  : Number of modules in the system
- **A**: Computation parameter
- **C**: Communication parameter



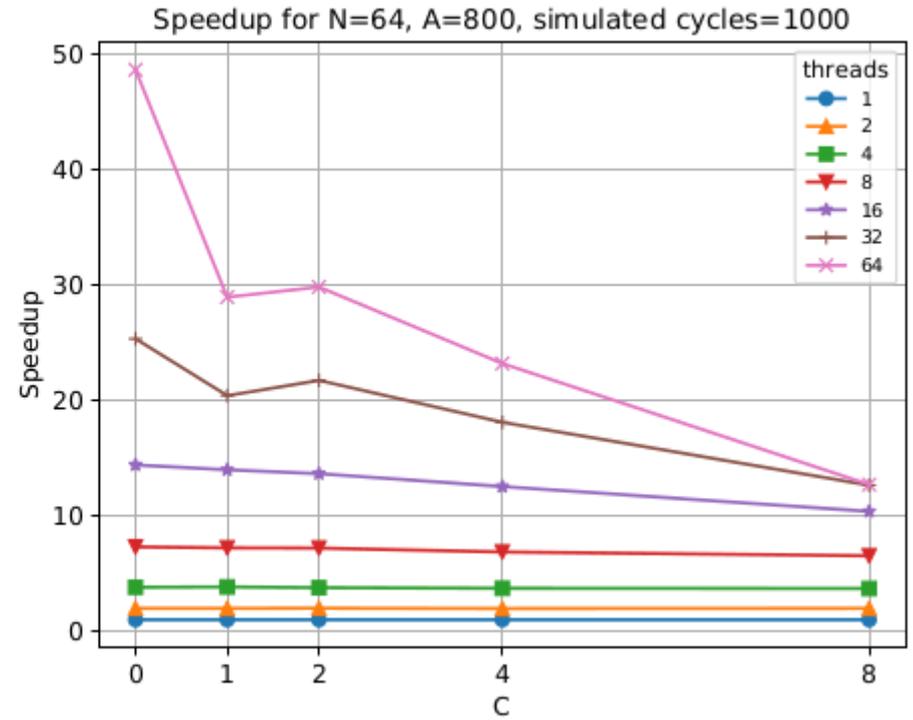
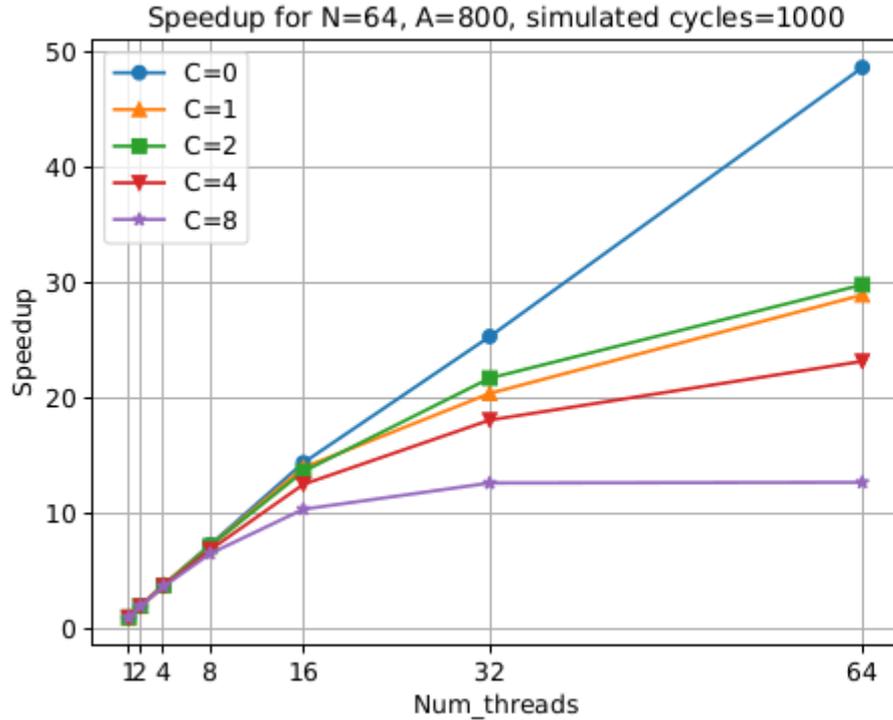
# Scalability (Model: Mesh with N modules Host:40-core rack server)



# Scalability (Model: Mesh with N modules Host:40-core rack server)

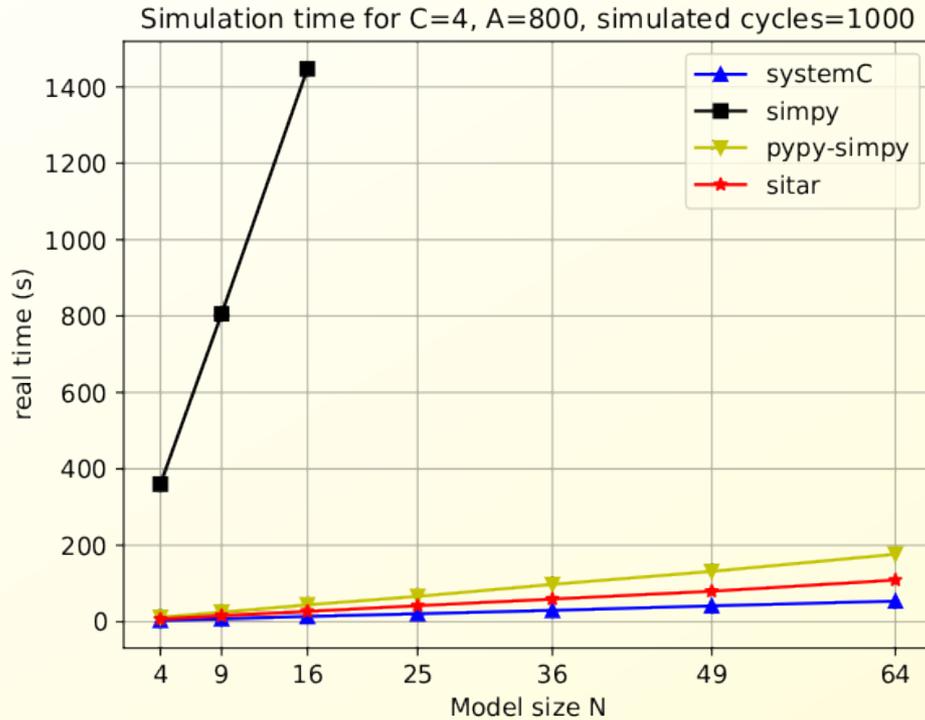


# Scalability (Model: Mesh with N modules Host:40-core rack server)

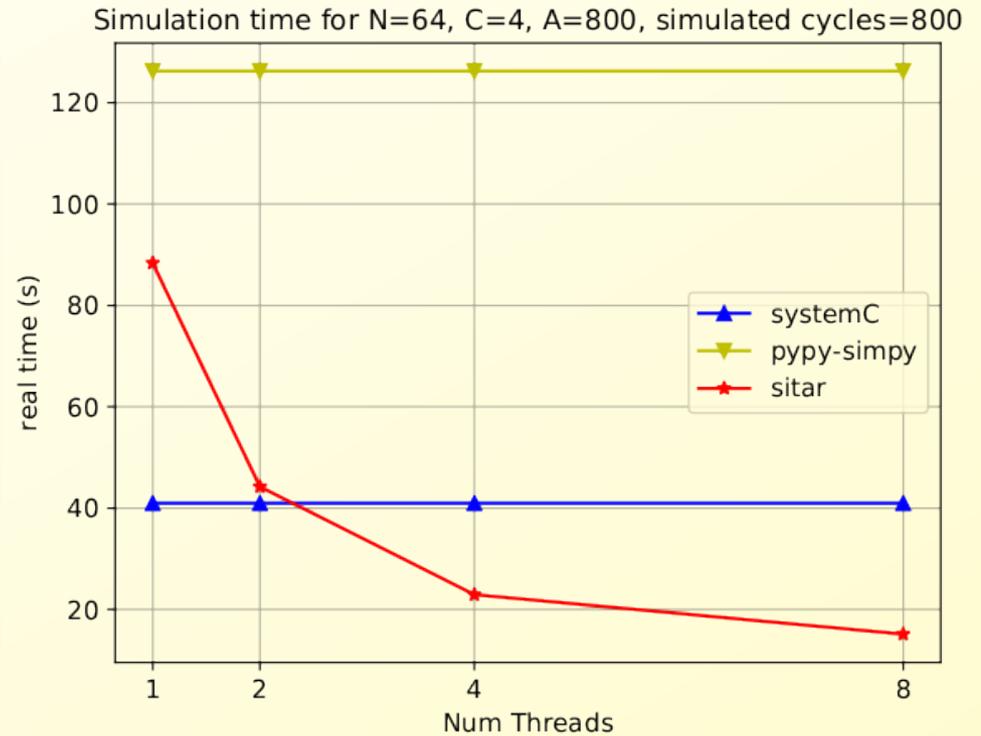


# Performance Comparison with SimPy and SystemC

## Single-threaded



## Multi-threaded Sitar



# Ongoing work

- Shared variable access to nets - performance takes a hit due to cache line transfer between threads.
- To be explored: MPI based communication and distributed system simulation
- Python port

# Conclusions

- Cycle-based simulation algorithm of Sitar enables fast, parallel simulations for discrete-time systems
- Sitar's modeling language provides a rich set of features for describing structure and behavior, along with logging support
- Sitar is suited to modeling discrete-time systems with a structural aspect (such as computer architecture models, networks etc)

# References

[1] **Sitar: A Cycle-based Discrete-Event Simulation Framework for Architecture Exploration** [Best Paper Award] Neha Karanjkar, Madhav Desai, *Proceedings of the 12th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2022)*, July 2022, Lisbon, Portugal

[2] **Efficient Parallel Simulation of Networked Synchronous Discrete-Event Systems**, Neha Karanjkar, Madhav Desai, Akhil Kushe, and Anish Natekar, Accepted for publication at WinterSim 2024, Orlando, Florida USA Dec 2024