Scalable system simulations for RISC-V architectures and performance analysis for machine learning workloads

Dive into the world of system-level simulations!

- Explore RISC-V modelling and workload representation using gem5+MLIR.
- Learn to scale your system simulations effortlessly with the power of SST.

This tutorial bridges cutting-edge open-source tools and techniques to empower your hardware-software co-design journey.

Debjyoti Bhattacharjee
Compute System Architecture (CSA)
imec, Belgium

Erwan Lenormand
Compute System Architecture (CSA)
imec, Belgium

Date: 5 January, 2025
Time: 4:00 PM
Duration: 90-Minute

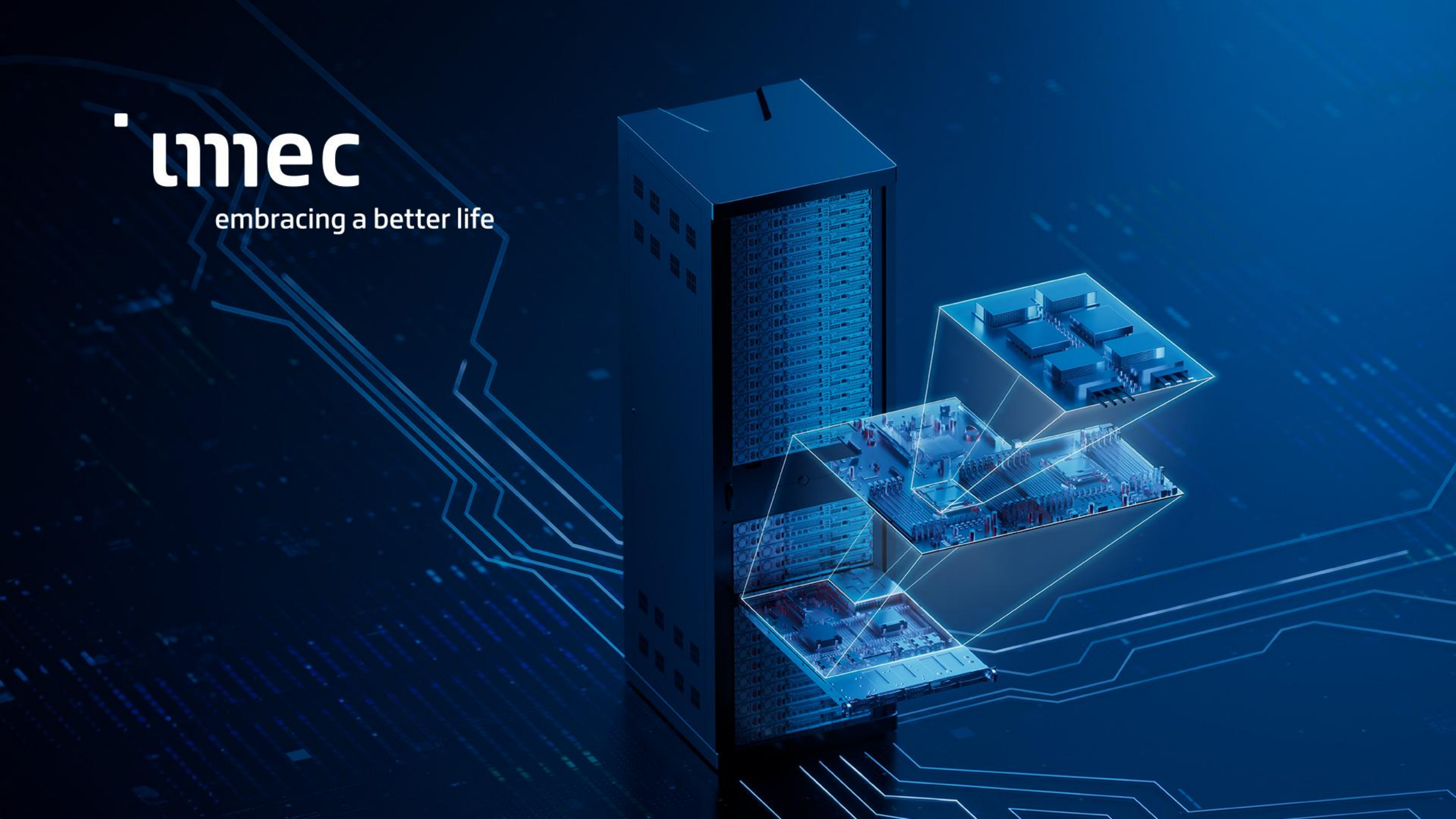
imec

Learn. Simulate. Scale.

VLSI DESIGN 2025 ES Embedded System

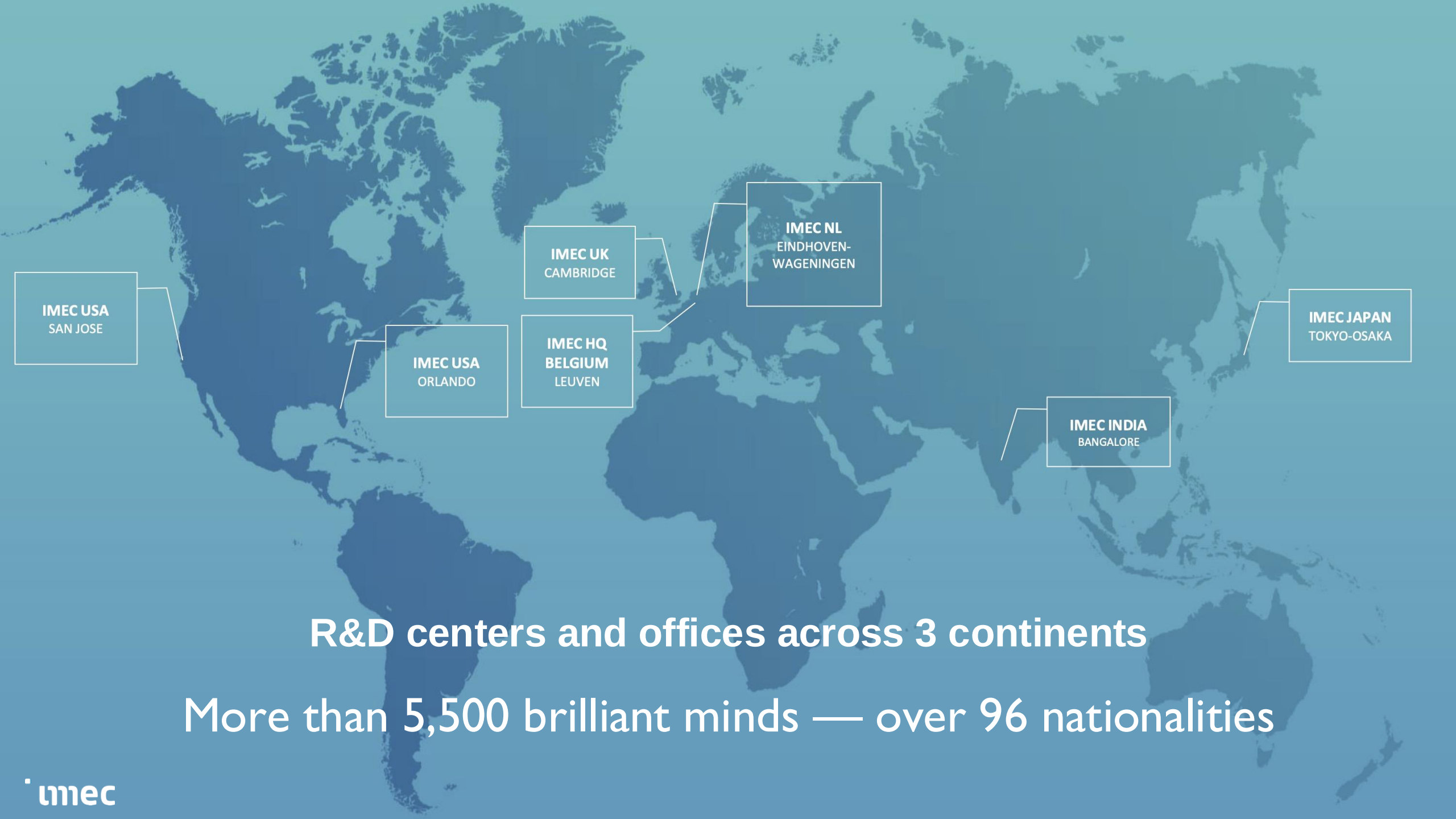
imec

embracing a better life




As a **world-leading R&D** hub, we aspire the impossible and aim for **disruptive innovation**.
We maximize societal impact by creating **smart sustainable solutions** that enhance **quality of life**.
At imec, we shape the future.

imec

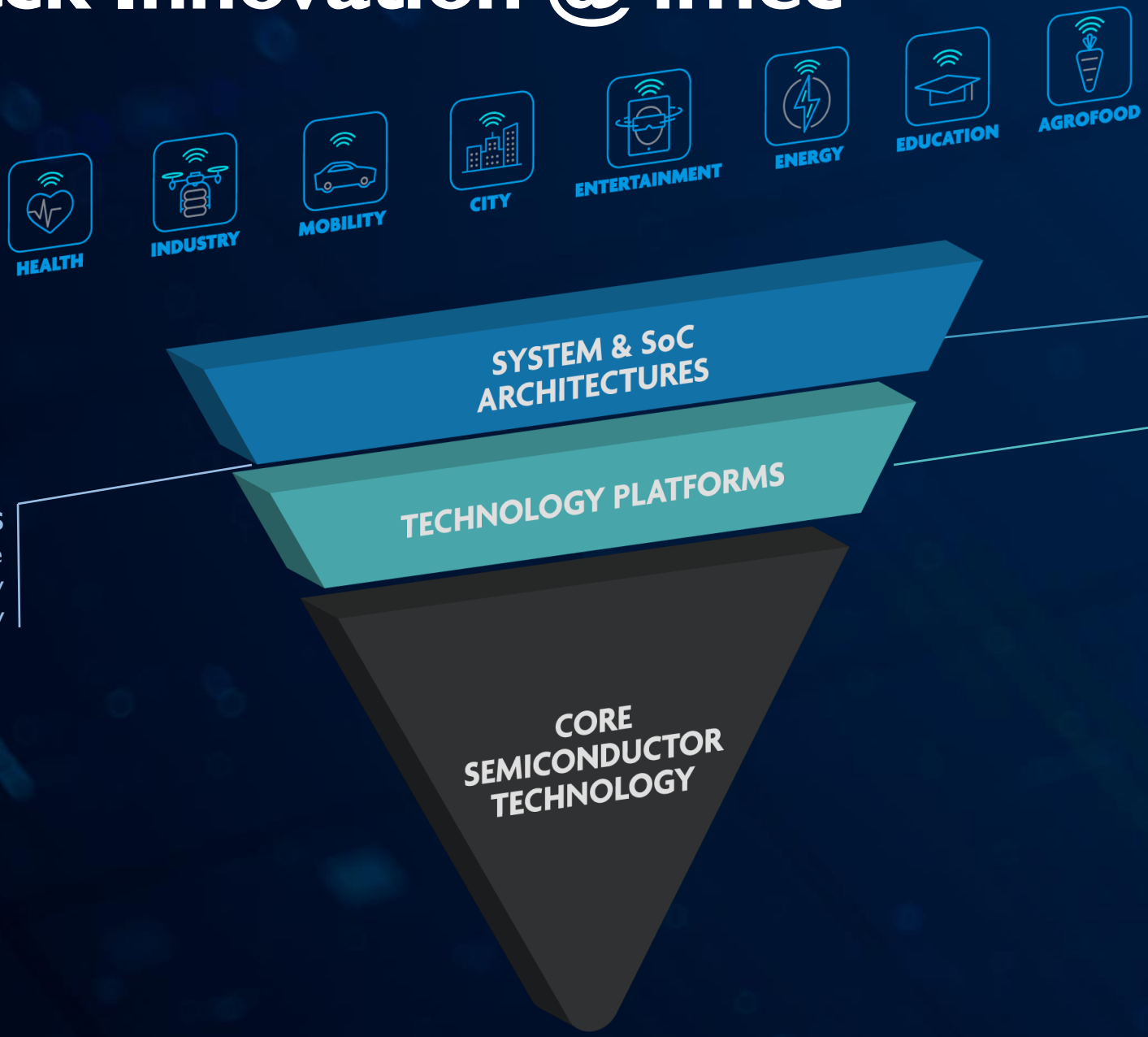


R&D centers and offices across 3 continents

More than 5,500 brilliant minds — over 96 nationalities

imec

Cross-Stack Innovation @ imec

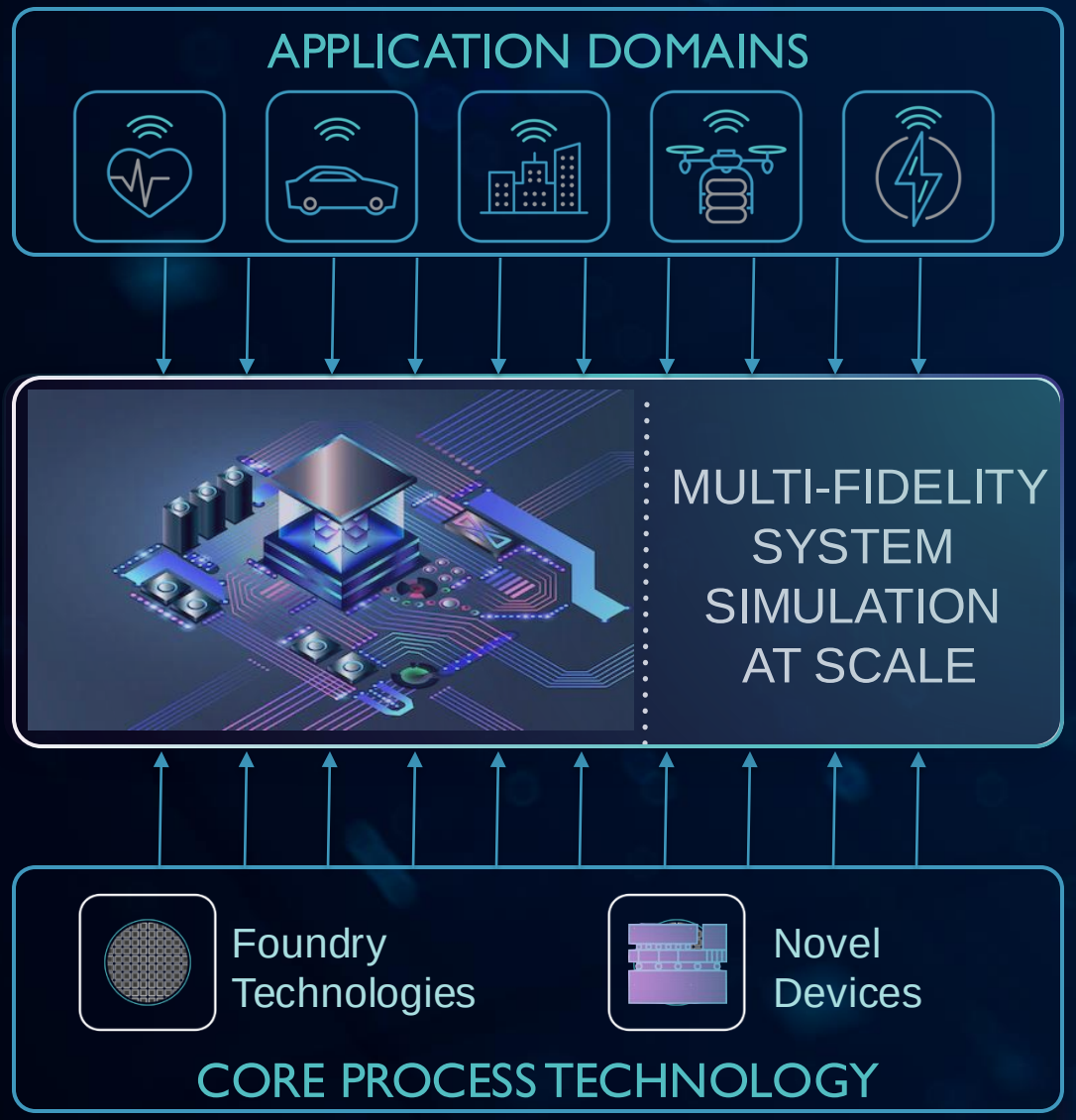


SYSTEM PLATFORMS
Compute System Architecture
Optical beamforming
DNA sequencing
Nanofluidics processor
RF technologies for (beyond) 5G
Hyperspectral imaging
…

DIGITAL TECHNOLOGIES
Artificial intelligence
Privacy / cybersecurity
(Beyond) 5G connectivity technology

TECHNOLOGY PLATFORMS
Logic CMOS
Memory
3D heterogeneous integration
Photonics
MEMS & microfluidics
Sensor platform
Flexible technologies

CSA : Areas of Expertise



- Domain-specific Architectures
- Performance and TCO Analysis
- HW Validation and Calibration
- Technology Impact Evaluation

Agenda

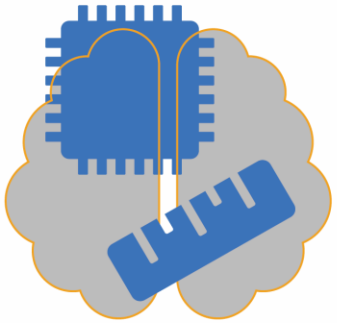
- **Introduction to System Simulation**
- Detailed Single Node simulation for ML workloads using gem5 x MLIR
- Scalable multi-node simulation using SST

ML: Machine Learning MLIR: Multi-Level Intermediate Representation SST: Structural Simulation Toolkit

System-Level Simulators

- Modern systems are increasingly complex
  - Requires accurate and scalable models to evaluate hardware-software interactions.
- Physical prototyping is expensive, time-consuming, and infeasible during early design stages.
- Simulators enable rapid exploration, performance analysis, and optimization of system architectures

gem5



SST

Astra-Sim

The Sparta Modeling Framework

AccelSim

https://www.gem5.org/ https://sst-simulator.org/
https://sparcians.github.io/map/index.html https://accel-sim.github.io/
https://scalesim-project.github.io/ https://astra-sim.github.io/

System-Level Simulators

Role of Simulation in Hardware-System Co-Design

- Use **system-level simulators** to model components (CPUs, memory, interconnects) and their interactions.
- Select/combine tools like **gem5** (detailed architectural modeling) and **SST** (scalable simulations) to analyse real-world scenarios
  - In most cases, a combination of multiple simulation tools are required to come to an "actionable" decision
- Use **representative benchmarks** and **design of experiments (DoE)** to ensure accuracy and scalability.

System-Level Simulators

Features and Limitations

- Typically allows modelling of a "hardware" component at high level of abstraction
  - Usually much less details than RTL
- Performance estimation and bottleneck analysis can be done
  - *Eg: Pipeline viewer to see stalls*
- Serves well to do "design space exploration" and answer "what-if" question(s)
  - *Eg: What if the processor clock could run at 30Ghz?*

How does "area", "energy" change with "technology"?

Direct estimation is not feasible.
Needs additional tools.

It does not say if it is feasible to design a CPU with 30Ghz clock.
RTL validation and possible full mapping to technology library would be necessary.

Design of Experiments (DoE)

Purpose
- Systematically explore design spaces to identify optimal configurations.

Key Metrics
- Define figures of merit (FoM)
- *Eg: Misses Per Kilo Instructions (MPKI) executed*

Parameter Selection
- Choose system variables to analyze.
- *Eg: cache size, memory latency*

Experiment Planning
- Use structured approaches (e.g., factorial design) to reduce simulation overhead.

Result Analysis
- Extract insights to understand potential bottlenecks and areas of improvement

Design of Experiments (DoE)

Choosing "Figure of Merit"

- Choosing the "figure of merit" is a very critical aspect in the DoE
- If a simulator cannot be used to report or derive an "figure of merit"
  - Choose a different simulator
  - Extend it or combine it with some other simulator → This is mostly what happens in practice!
- Often a combination of multiple metrics might be interesting
  - *Eg: Area-Delay Product, Throughput/mm^2*

Design of Experiments (DoE)

Benchmarks for Evaluation

- Choice of benchmarks is as important as choice of "FoM"
  - *Eg: If you want to evaluate a multi-node system, MPI workloads might be interesting*
- It always helps to use "well-accepted" benchmarks to compare across different systems of similar kind
  - *Eg: SPEC CPU® 2017, LINPACK benchmarks, etc.*

Benchmarks need to be represented/compiled in a way that the simulator/system can ingest!

Agenda

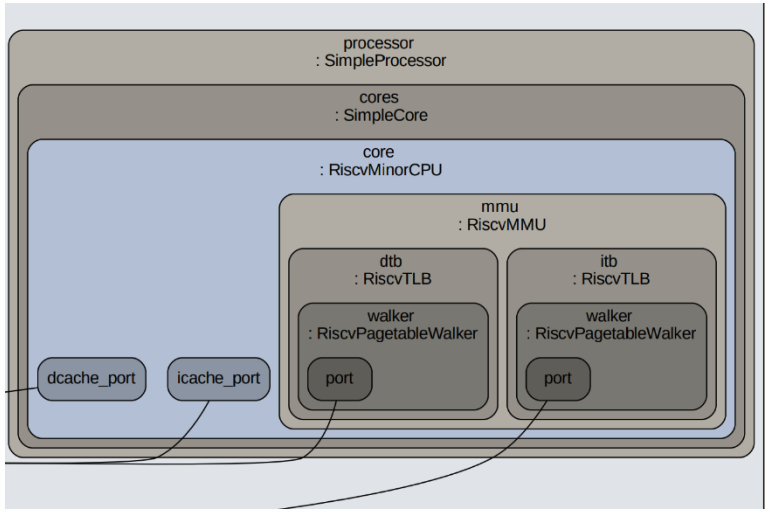
- Introduction to System Simulation
- **Detailed Single Node simulation for ML workloads using gem5 x MLIR**
- Scalable multi-node simulation using SST

ML: Machine Learning MLIR: Multi-Level Intermediate Representation SST: Structural Simulation Toolkit

gem5

A brief introduction

- Modular discrete event driven computer system simulator platform
  - Used widely in academia as well as industry
- Describes a system in terms of components
  - Generally expressed in C++
- Components are connected by means of Python script
  - Each component is parameterized as well
  - Statistics per component is reported



A part of a "simulated system"

gem5

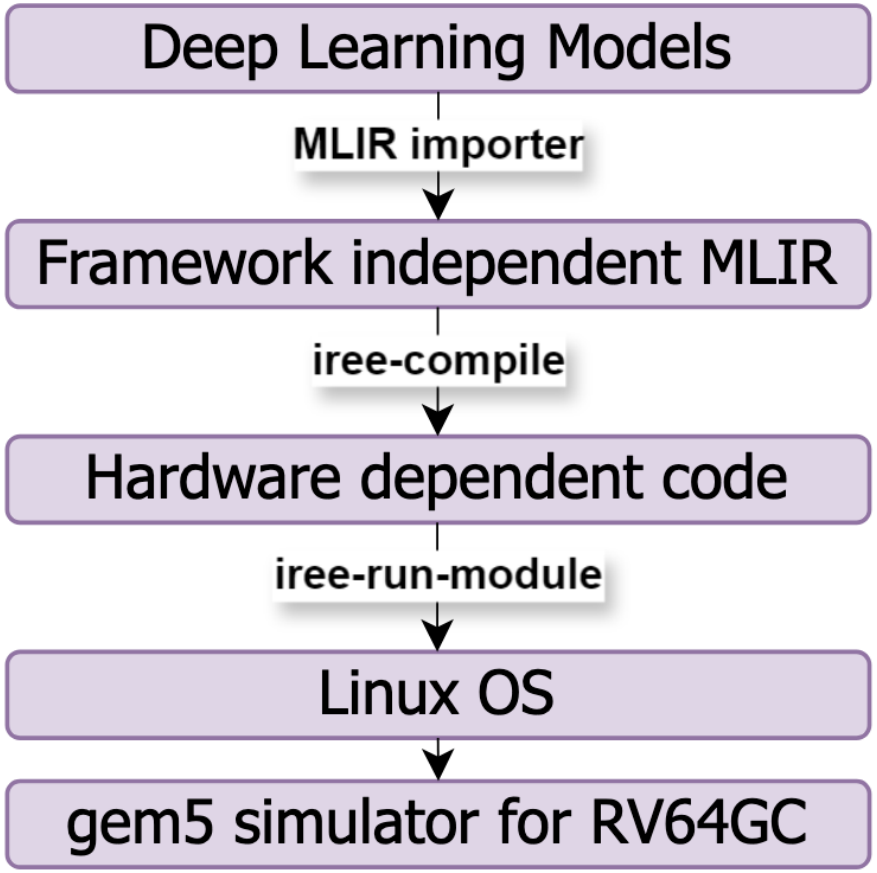
Representing a RISC-V System

- Gem5 core component micro-architecture is separate from the ISA
  - *Eg: MinorCPU models an in-order architecture →*
    Can be combined with "RV64GC" or "RV64IMA"
- Different cache hierarchies and cache coherency protocols can be modelled as well
- Much more possible! → www.gem5.org

Attribute	Type/version
Core Type	MinorCPU, O3CPU
Core Freq.	2 GHz
L1 Cache	64KB, 4-way
L2 Cache	8MB, 4-way
DRAM Type	simpleMem
DRAM Size	3GB
DRAM Freq.	1 GHz

Mapping Workloads to System Simulations

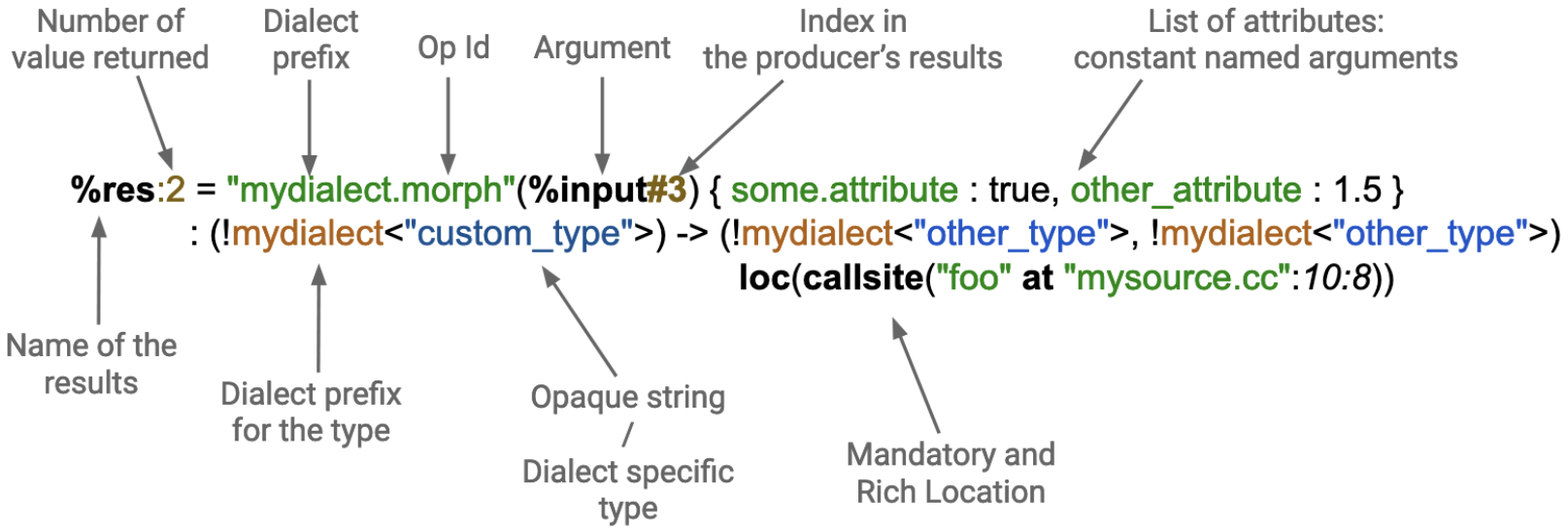
- We are considering Machine Learning Inference task as workload in this tutorial
- **Workload Abstraction:** MLIR provides a flexible intermediate representation for machine learning workloads, enabling optimization across hardware targets.
- **Exporting from ML Frameworks:** MLIR can be generated from popular ML frameworks like TensorFlow, PyTorch, and JAX using frontend integrations.
  - *Eg: tflite-to-mlir converter is used in this tutorial*



Mapping Workloads to System Simulations

Operations, Not Instructions

- No predefined set of instructions
- Operations are like “opaque functions” to MLIR



*https://llvm.org/devmtg/2019-04/slides/Tutorial-AminiVasilacheZinenko-MLIR.pdf

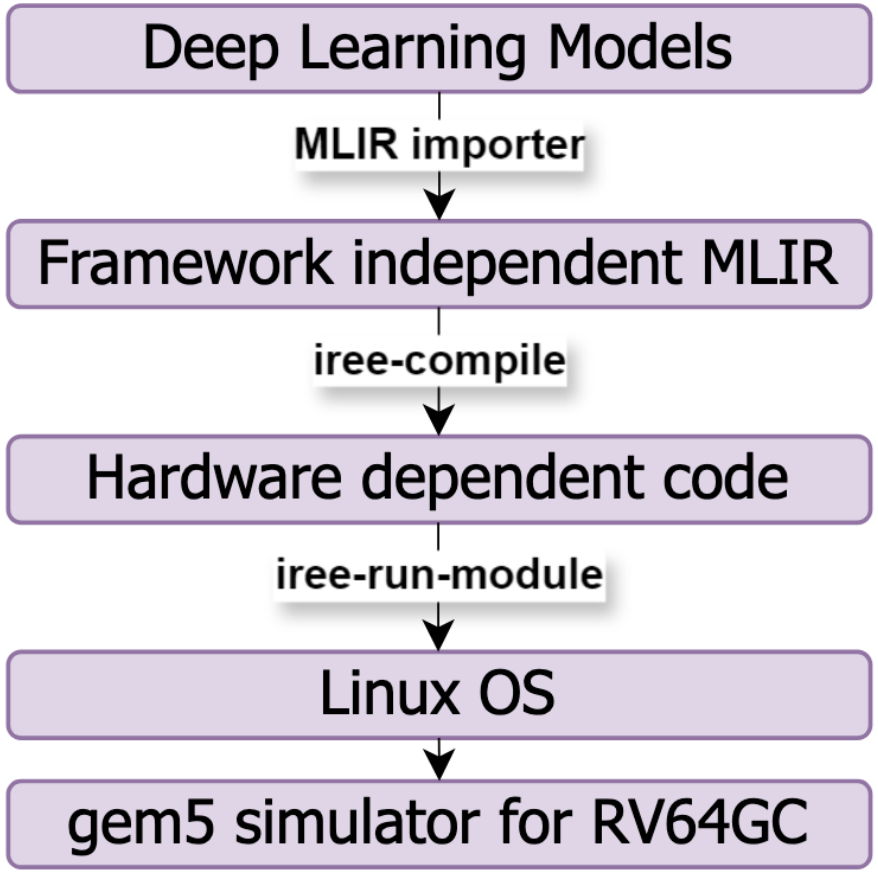
Mapping Workloads to System Simulations

```
module @pmap__unnamed_wrapped_function_ attributes {mhlo.num_partitions = 1 : i32, mhlo.num_replicas = 4 : i32} {
func.func public @main(%arg0: tensor<i32> {mhlo.is_same_data_across_replicas = true}, %arg1: tensor<64xf32> ...
%1 = stablehlo.reshape %arg433 : (tensor<1x64xi32>) -> tensor<64xi32>
%c = stablehlo.constant dense<0> : tensor<i32>
%2 = stablehlo.subtract %arg0, %c : tensor<i32>
%c_0 = stablehlo.constant dense<0> : tensor<i32>
%c_1 = stablehlo.constant dense<1000> : tensor<i32>
%3 = call @clip(%2, %c_0, %c_1) : (tensor<i32>, tensor<i32>, tensor<i32>) -> tensor<i32>
%4 = stablehlo.convert %3 : (tensor<i32>) -> tensor<f32>
%cst = stablehlo.constant dense<1.000000e+03> : tensor<f32>
%5 = stablehlo.divide %4, %cst : tensor<f32>
%cst_2 = stablehlo.constant dense<1.000000e+00> : tensor<f32>
%6 = stablehlo.subtract %cst_2, %5 : tensor<f32>
%7 = call @integer_pow(%6) : (tensor<f32>) -> tensor<f32>
...
```

*https://llvm.org/devmtg/2019-04/slides/Tutorial-AminiVasilacheZinenko-MLIR.pdf

Mapping Workloads to System Simulations

- **Dialect Usage**: Leverage MLIR's dialects to model ML computations effectively.
  - *Eg: Stablehlo*
- **IREE Compilation***: Use IREE to lower MLIR representations into RISC-V binaries (depending on chosen ISA)
  - Applies multiple MLIR's transformation passes to optimize workloads
  - Also, embeds scheduling information
- **End-to-End Flow:** Execute the generated code using *iree-run-module* on the simulated system



*https://iree.dev/

Experiments

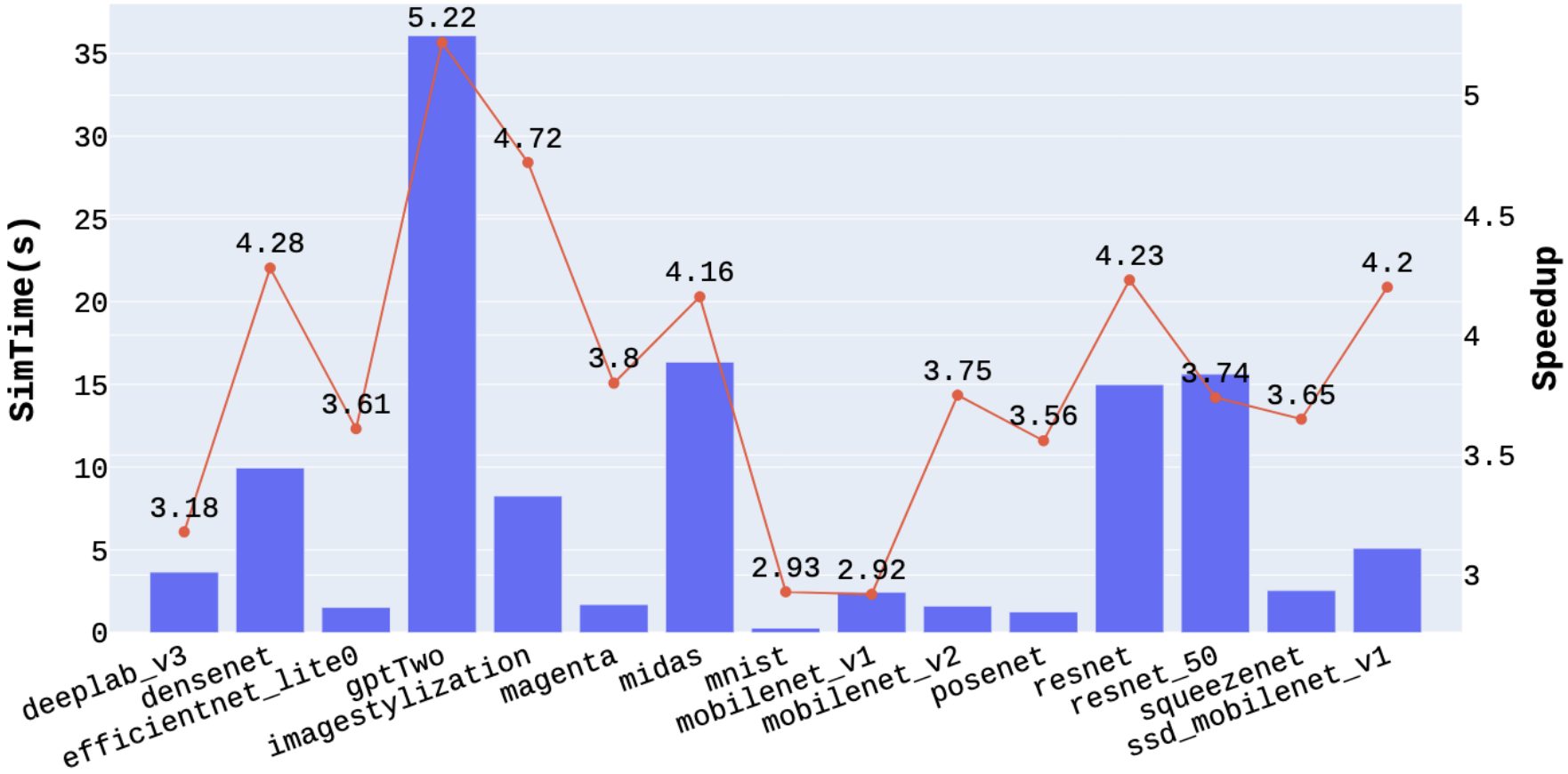
Notes for Participants

- The scripts for running the experiments on gem5 using the proposed flow is available on github
- Docker environment is provided
- Try it out yourself
  - File issues on the github if something does not work!
  - Reach out to us!

https://github.com/CSA-infra/RISCV-Scalable-Simulation-tutorial

Exp1: Simulating a RISC-V Core with gem5

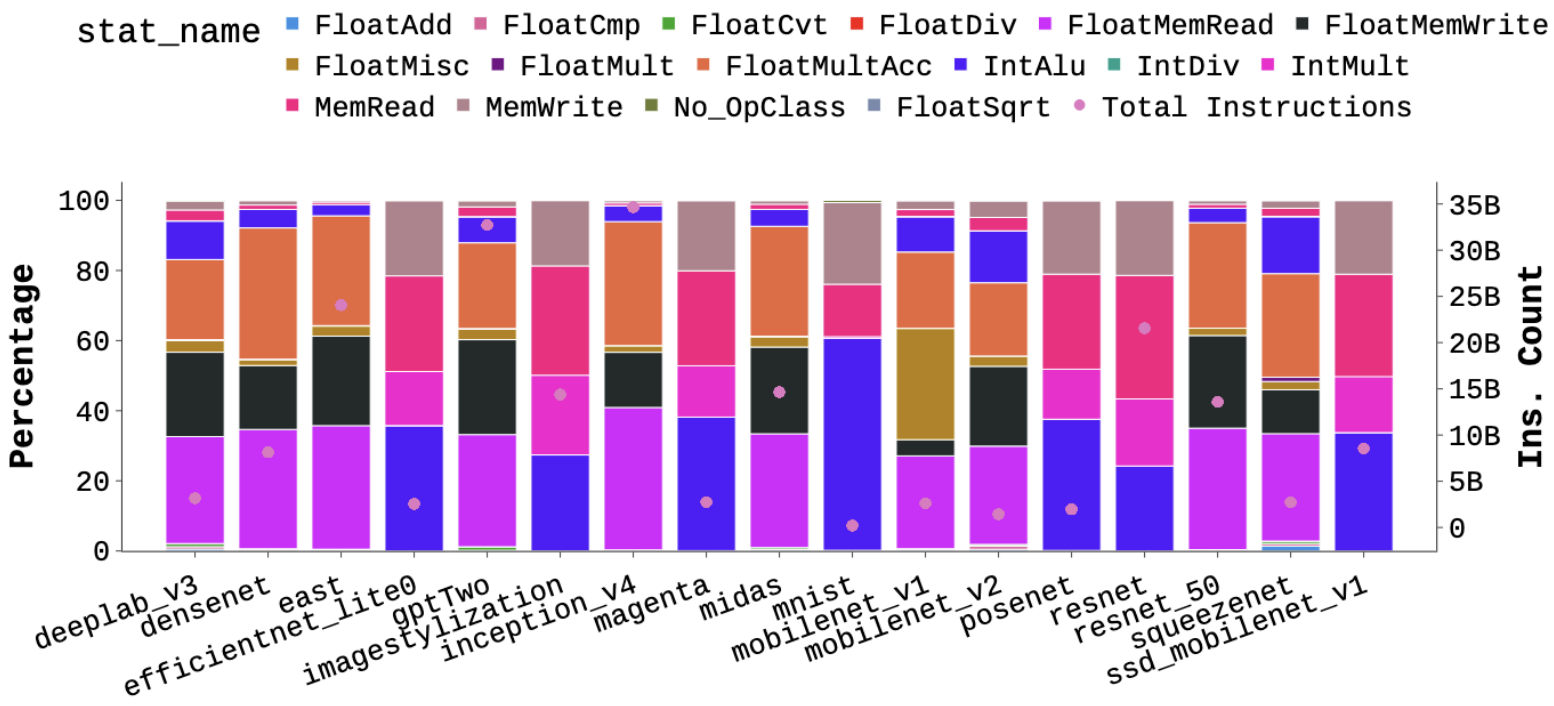
- Performance of the workloads running on
  - *in-order CPU* (Minor CPU)
  - Speedup of *Out-of-order(O3) CPU* vs *in-order CPU*



https://arxiv.org/pdf/2405.15380

Exp2: Understanding the instruction mix of workloads

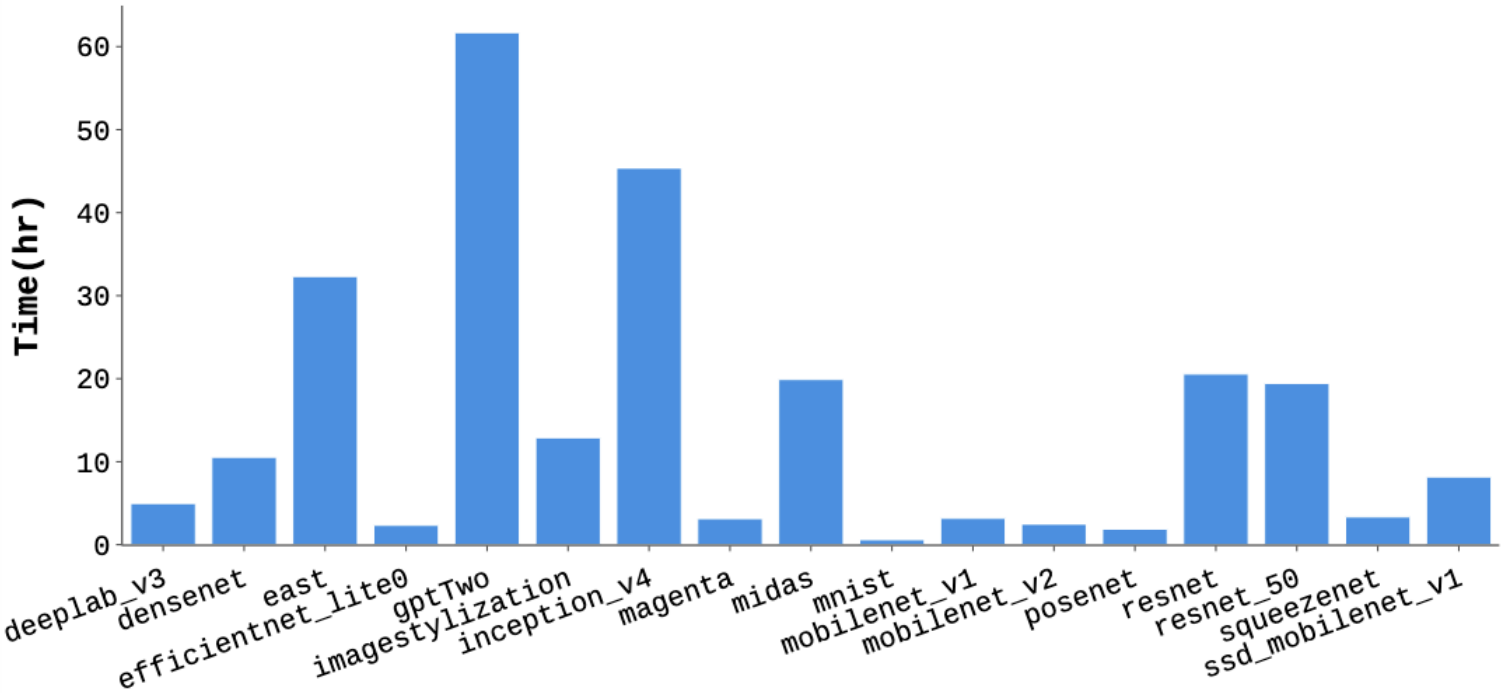
- Each run of gem5 dumps the different types of instructions executed by the CPU
- Can you spot which ones are "integer"/ "quantized" models by looking at this plot?



https://arxiv.org/pdf/2405.15380

Exp3: How long does it take to simulate each workload?

- Gem5 out-of-the-box is a "single-threaded" simulator
- For large workloads
  gptTwo > 2 days to simulate
- For even larger LLMs, this would grow intractable!



https://arxiv.org/pdf/2405.15380

Scaling out the simulation is essential to model and analyse the complex large, interconnected systems efficiently

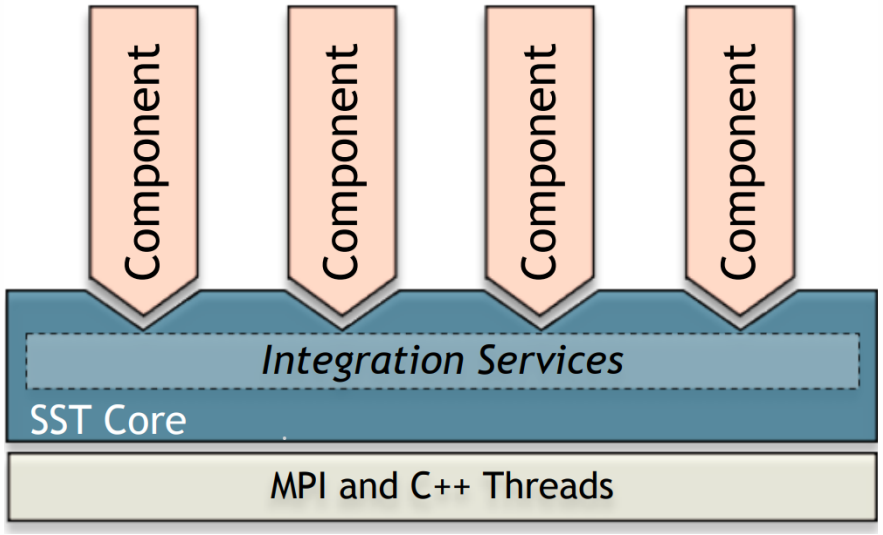
Agenda

- Introduction to System Simulation
- Detailed Single Node simulation for ML workloads using gem5 x MLIR
- **Scalable multi-node simulation using SST**

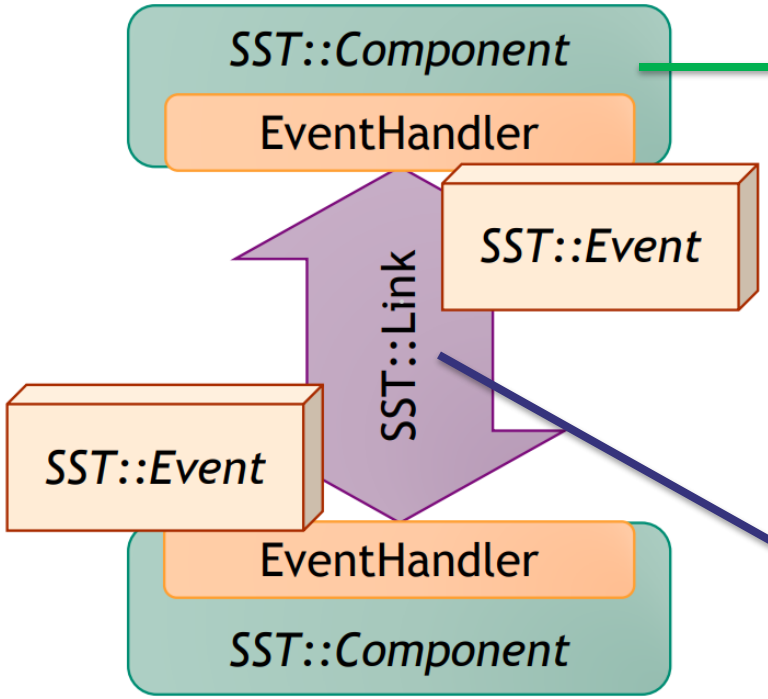
ML: Machine Learning MLIR: Multi-Level Intermediate Representation SST: Structural Simulation Toolkit

Introduction to Structural Simulation Toolkit (SST):

- SST is a Parallel discrete-Event Simulator Framework
  - Developed by Sandia National Laboratories
  - Uses MPI for parallelization: **Allows large scale simulation**
- Divided into two projects:
  - **SST-core framework**:
    - The backbone of the simulator
    - Provides utilities and interfaces for components
    - **You don't need to edit it**
  - **SST-Elements Libraries**:
    - Libraries of components that model compute system
    - You can add new components or libraries
    - **More information on** https://sst-simulator.org
- Components are modeled in C++
- Simulations are built with python script



Introduction to Structural Simulation Toolkit (SST): Building a system



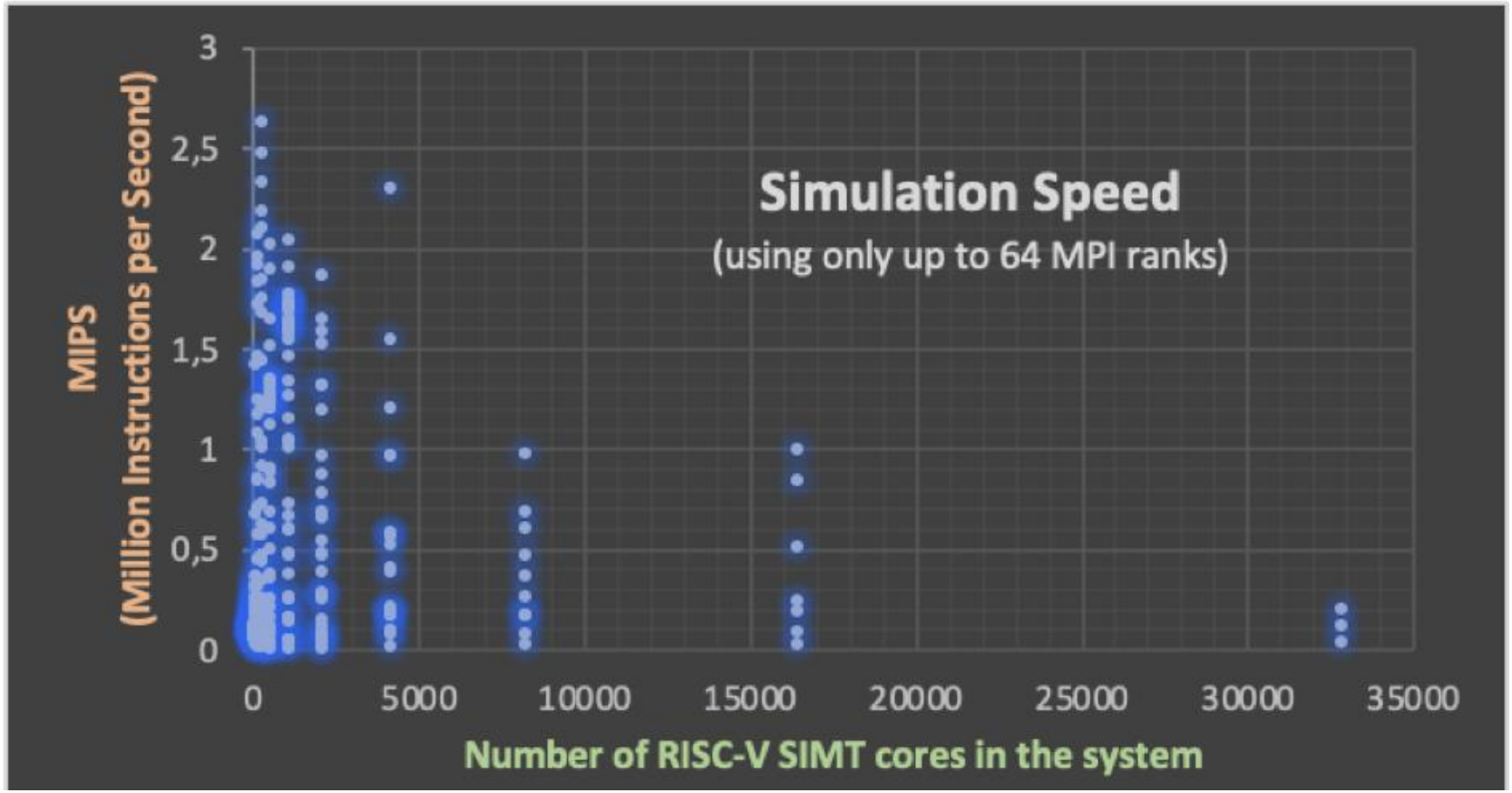

```
### Create the components
component0 = sst.Component("c0", "simpleElementExample.example0")
component1 = sst.Component("c1", "simpleElementExample.example0")
                                   Library.Component


### Parameterize the components.
params = {
        "eventsToSend" : 50,
        "eventSize" : 32
}
component0.addParams(params)
component1.addParams(params)



# Link the components via their 'port' ports
link = sst.Link("component_link")
link.connect( (component0, "port", "1ns"), (component1, "port", "1ns") )
```

Parameters and port names must match information provided by sst-info

Source: A-SST Initial Specification

Python script snippet

Introduction to Structural Simulation Toolkit (SST):



SST is used in imec to perform scale-out simulation!

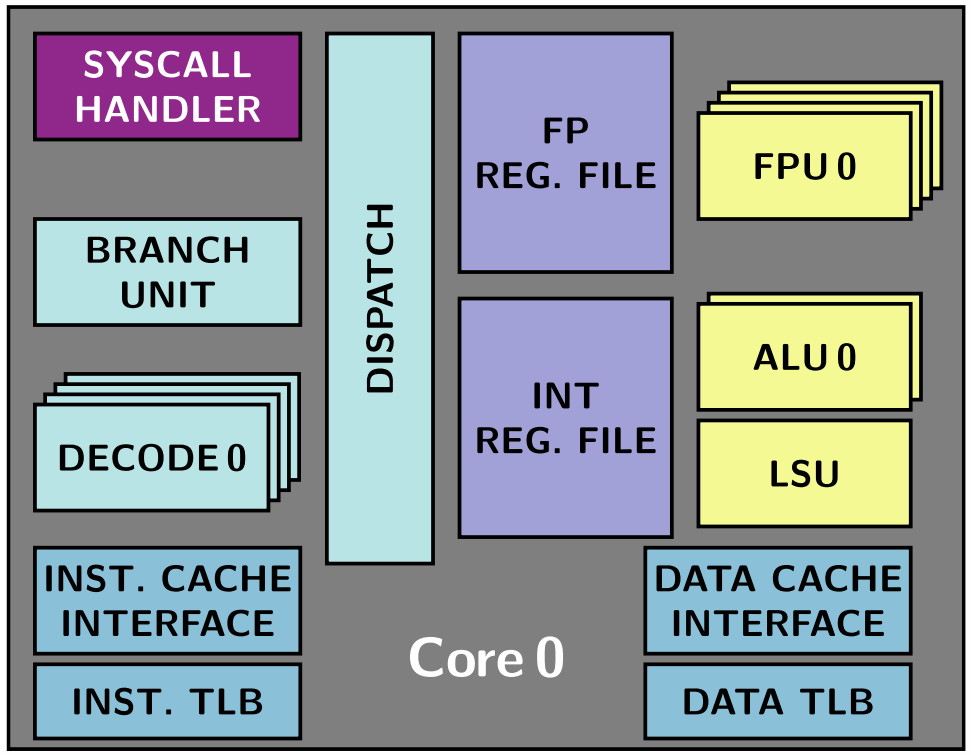
Experiments

Notes for Participants

- The scripts for running the experiments with SST using the proposed flow is available on github
- Instructions to build SST-core and SST-elements are provided in *external/INSTALL.rst*
- Workload binaries are already compiled!
- Try it out yourself
  - File issues on the github if something does not work!
  - Reach out to us!

https://github.com/CSA-infra/RISCV-Scalable-Simulation-tutorial

Presentation of the system under exploration: Single-node system

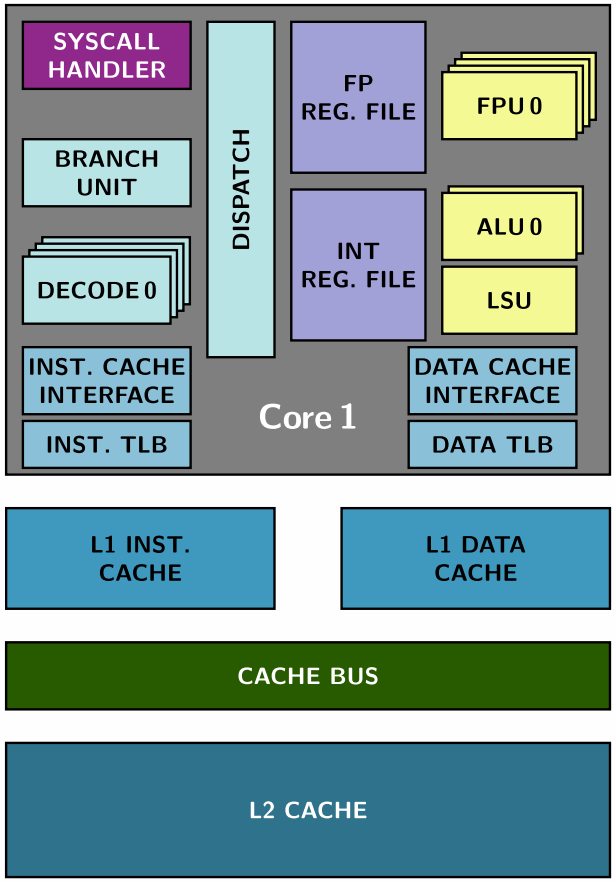


RISC-V 64-bit CPU core

The CPU core is simulated with ***Vanadis*** model:

- Vanadis is a processor with **detailed pipeline** model
- The ISA is configurable: MIPS32 or **RISCV64**
- Supports **multi-threading**
- **Does not support vector extensions**
- **Widely configurable:**
  - Register file size
  - Number of instructions issued per cycle
  - Number of ALU and FPU
  - Latencies of ALU and FPU operations
  - Etc ...

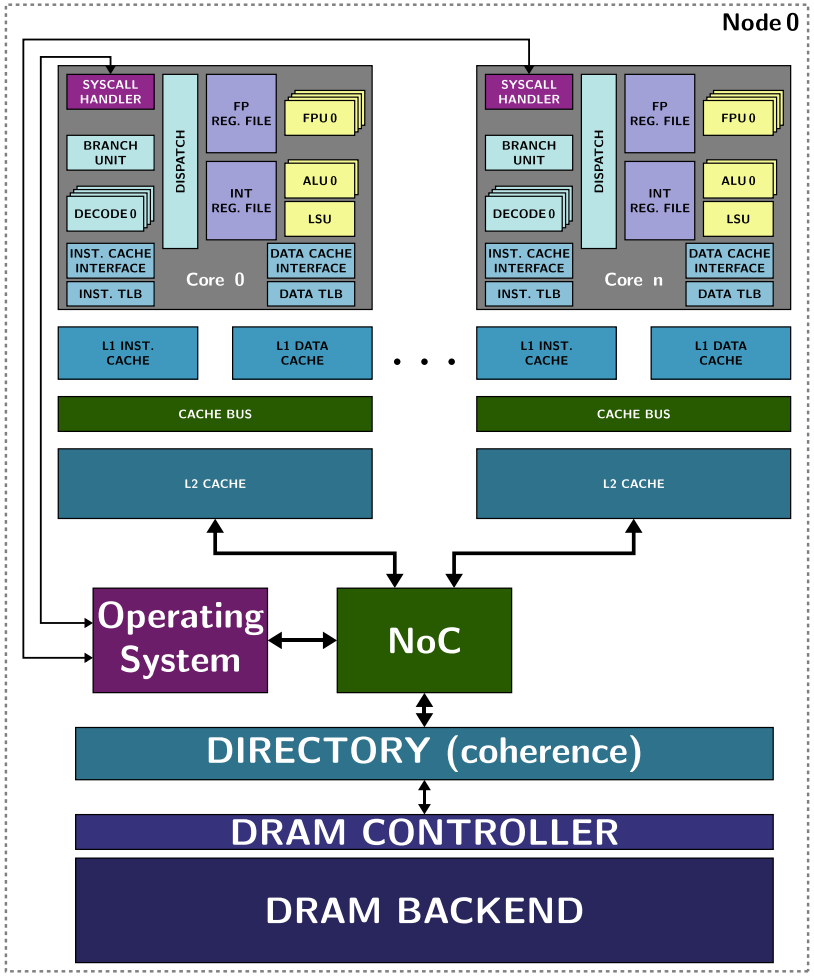
Presentation of the system under exploration: Single-node system



CPU complex

- **Each CPU core is attached to cache subsystem**:
  - Private L1 Instruction cache with a prefetcher
  - Private L1 Data cache with a prefetcher
  - Shared L2 cache (inclusive)
  - MESI cache coherence protocol
  - A memory bus interconnect the caches
- **Every components are defined in memHierarchy**
- **Memory components are configurable**:
  - Capacity (cache & buffers)
  - Latency (ports & caches)
  - Bandwidth (link width and number of requests per cycle)

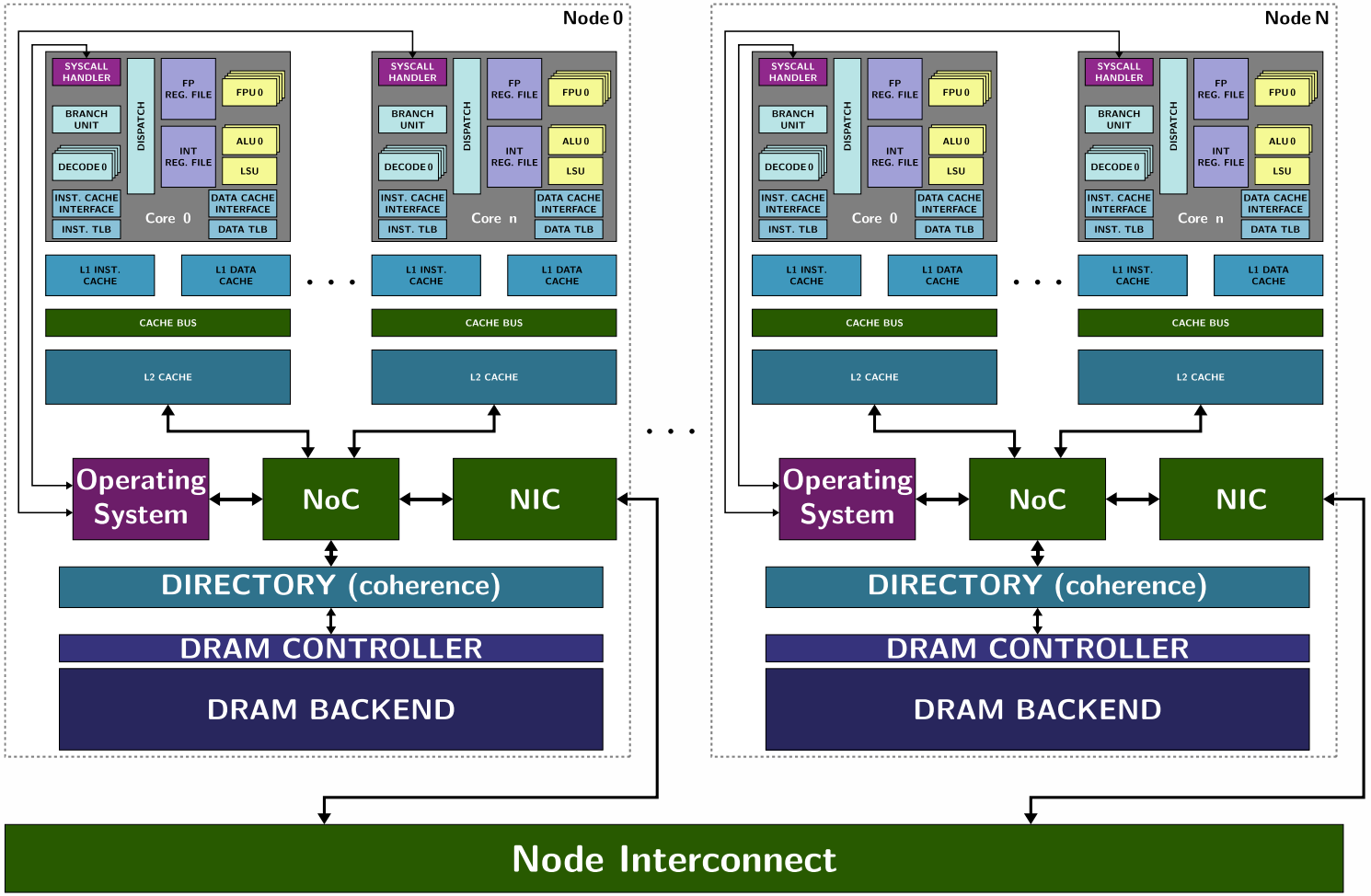
Presentation of the system under exploration: Single-node system



A single-node system is built by aggregating multiple CPU complex with a Network-on-Chip:

- Integrates an **operating system** providing the minimal services to run applications including virtual memory:
  - Emulated with an SST-elements component:
    - Fast syscall emulation
    - Not extendable
- Integrates a **DRAM** attached to **a directory** which manages the coherency:
  - Many backend can be instantiated:
    - Naive memory model applying constant latency (simpleMemory)
    - External DRAM simulator accurately reproducing latencies (Ramulator from CMU-Safari)
- **A node can run multi-threaded applications**
- **Scaling-up compute power by increasing the number of CPU cores**

Presentation of the system under exploration: Multi-node system

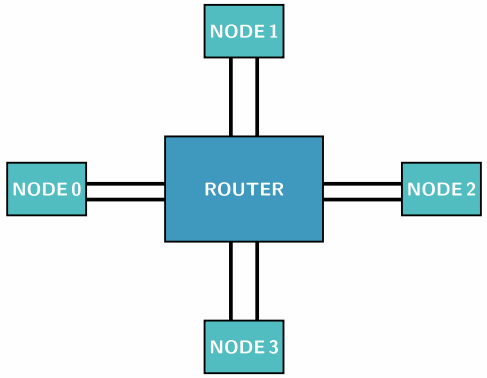


A Multi-node system is built by aggregating node with a node interconnect:

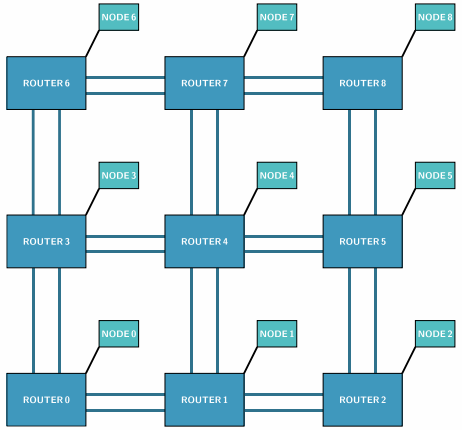
- A NIC is attached to each node:
  - **Allows to run MPI applications**
  - **Requires to edit the MPI library**
- **Many interconnect can be instantiated**:
  - Allows to do studies on network topologies

Scaling-out compute power by increasing the number of nodes

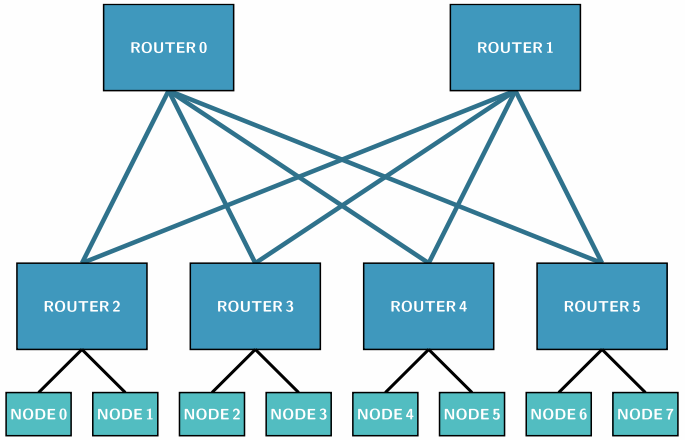
Presentation of the workload under evaluation: Merlin topologies



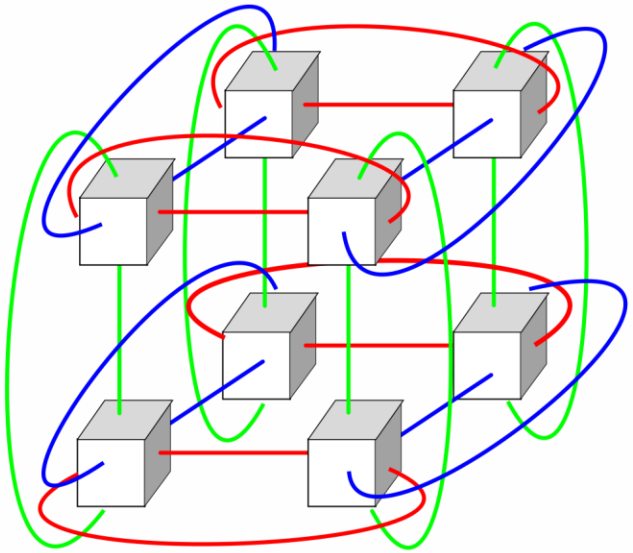
Single router



2D Mesh



Fat tree



2x2x2 Torus

(from: Wikipedia/Torus interconnect)

SST Merlin allows to explore many topologies with minimal effort:

Includes also:

- **Dragon fly topology**
- **Hyper X topology**

Workload under evaluation: Multi-Head Attention

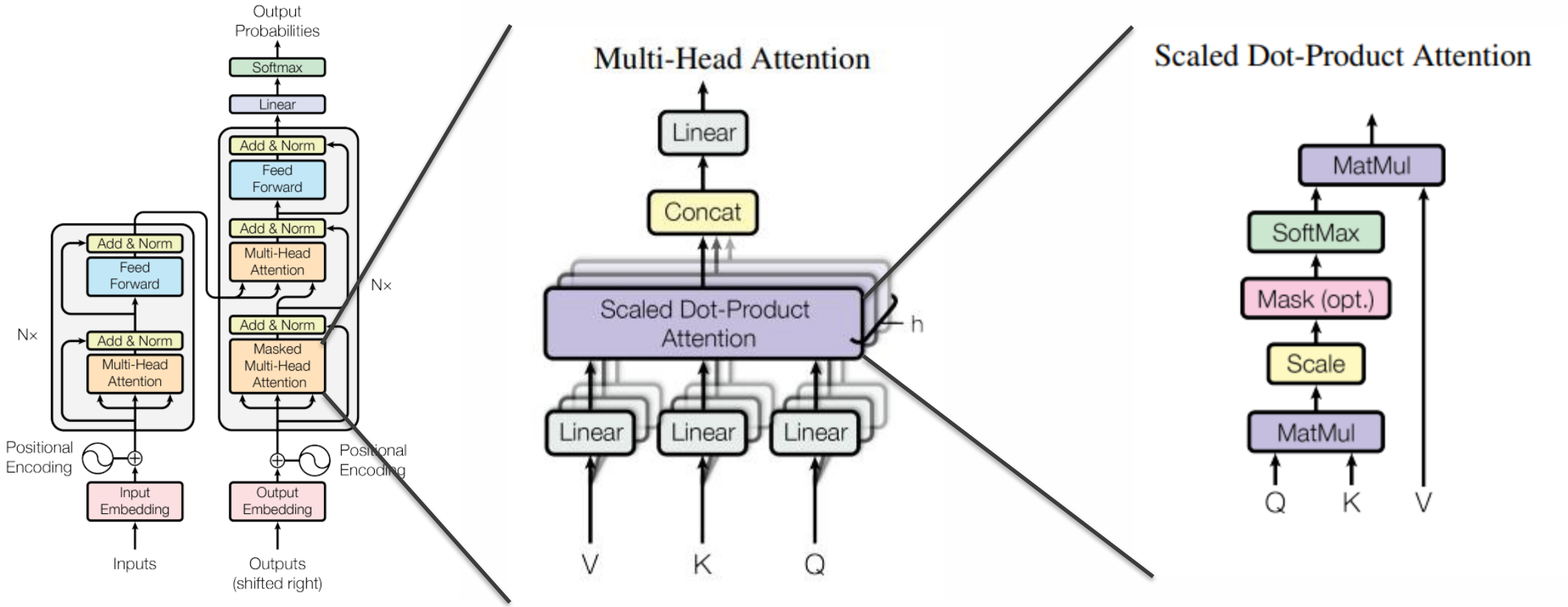
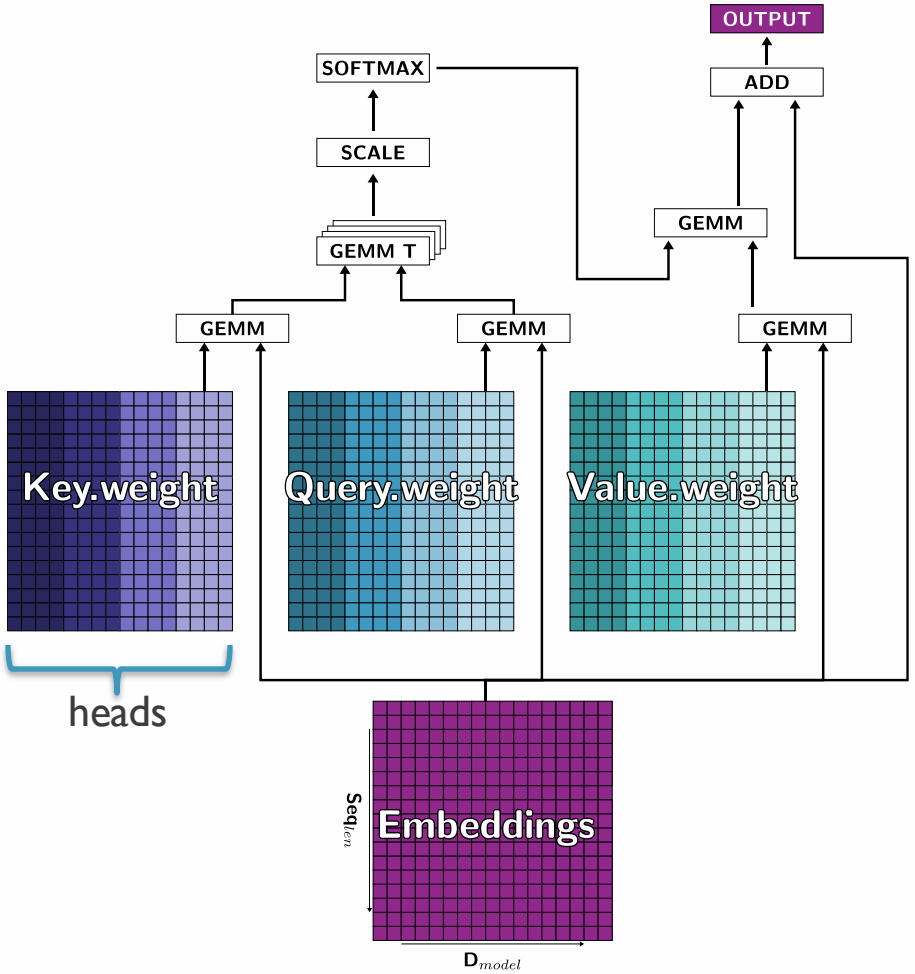


Illustration from: Attention Is All You Need

Demo 1: Single-node simulation with an OpenMP application



- Seq_{Len}, D_{model} and ***heads*** are passed as arguments of the main function and defined the number of FP operations (FLOP):
  - Q,K,V, and ATTNout GEMM: each requires $2 \times Seq_{len} \times D_{model} \times D_{model}$ FLOP
  - KQ and KQV GEMM: each require $2 \times Seq_{len} \times Seq_{len} \times D_{model}$ FLOP
  - The scale kernel requires $heads \times Seq_{len} \times Seq_{len}$ FLOP
  - The softmax kernel requires $7 \times heads \times Seq_{len} \times Seq_{len}$ FLOP
  - The ADD kernel requires $Seq_{len} \times D_{model}$ FLOP
  - **Total:** $\boldsymbol{Seq_{len} \times (D_{model} \times (8 \times D_{model} + 4 \times Seq_{len} + 1) + 8 \times heads \times Seq_{len})}$
- **Each compute kernel is parallelized with OpenMP directives**
- Matrix-Matrix multiplication is the heaviest workload:
  - **Tiled implementation** to leverage cache hierarchy and minimize data movement

Demo 1: Single-node simulation with an OpenMP application

1st experiment: GEMM tile size

- 4 binaries are provided for this experiment:
  - mha_OMP_8 which uses **8x8 tiles**
  - mha_OMP_16 which uses **16x16 tiles**
  - mha_OMP_32 which uses **32x32 tiles**
  - mha_OMP_64 which uses **64x64 tiles**
- Run the two binaries and analyze the performance:
  - Run `**sst scale_up.py**` in *demo/sst/system* folder
  - You need to pass the correct **exe (--exe path)** as argument of the script
- **What can explain the difference?**
  - You can use the generated statistics to verify the explanation

2nd experiment: simulation scaling

Let's explore the scalability of the **simulated system** and of the **simulation**:

- Make sure to use **the most efficient binary**
- You can vary the number of simulated **threads** and **processors**
  - **--num_threads_per_cpu NUM**
  - **--num_cpu_per_node NUM**
- You can vary the number of simulation threads (i.e. parallelism of the simulation) using --num-threads= option:
  - **sst --num-threads=4 scale_up.py**

Demo 1: Single-node simulation with an OpenMP application

1st experiment: GEMM tile size

TILE SIZE	SIMULATED TIME		MHA TIME		Number of instructions issued			Number of loads executed		Number of branches	
	ms	Normalized	ms	Normalized	Millions of instructions	Normalized	Balance	Millions of loads	Normalized	Millions of branches	Normalized
8x8	10.83	1.11	10.31	1.12	55.23	1.22	50%/50%	10.92	1.06	6.58	1.11
16x16	10.08	1.03	9.57	1.04	48.30	1.07	50%/50%	10.50	1.02	6.12	1.04
32x32	9.76	1.00	9.24	1.00	45.21	1.00	50%/50%	10.34	1.00	5.91	1.00
64x64	12.86	1.32	12.34	1.34	54.61	1.21	54%/46%	12.44	1.20	10.15	1.72

Experiment with 2 CPU and 2 threads each. Seq_{len} = 64, D_{model},=128, heads = 8.

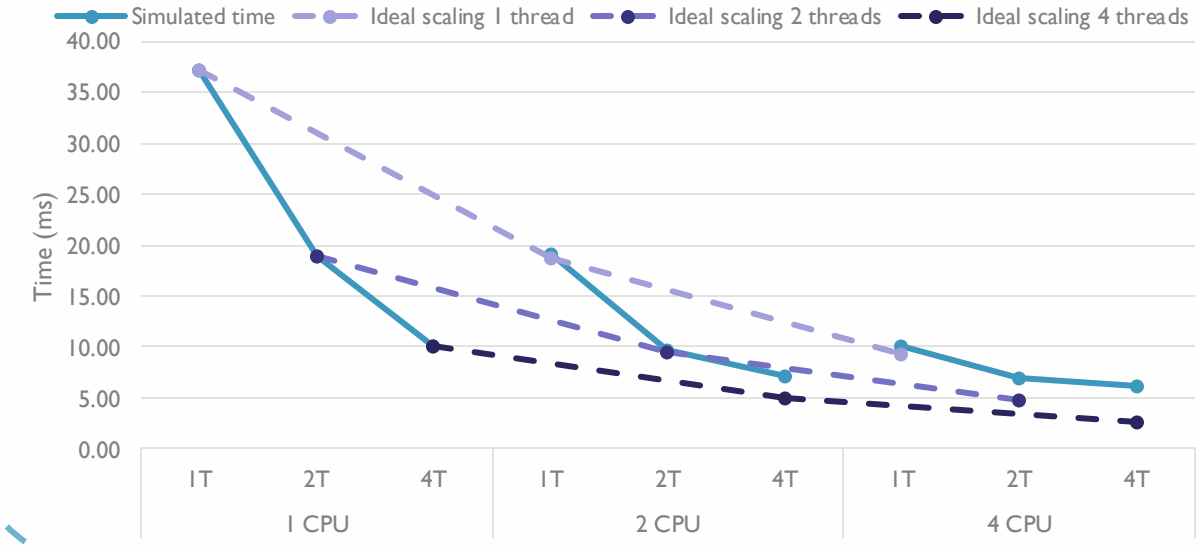
- Larger tiles improve the performance by increasing the arithmetic intensity (less loads and branches)
- Too large tiles lead to workload imbalance

Demo 1: Single-node simulation with an OpenMP application

2nd experiment: simulation scaling

	1 CPU			2 CPU			4 CPU		
Number of simulated threads	1T	2T	4T	1T	2T	4T	1T	2T	4T
Simulation time (ms)	37.26	18.84	10.13	19.21	9.76	7.06	10.09	6.88	6.14

Scaling of the simulated system with Seq_{len} = 64, D_{model}=128, heads = 8.



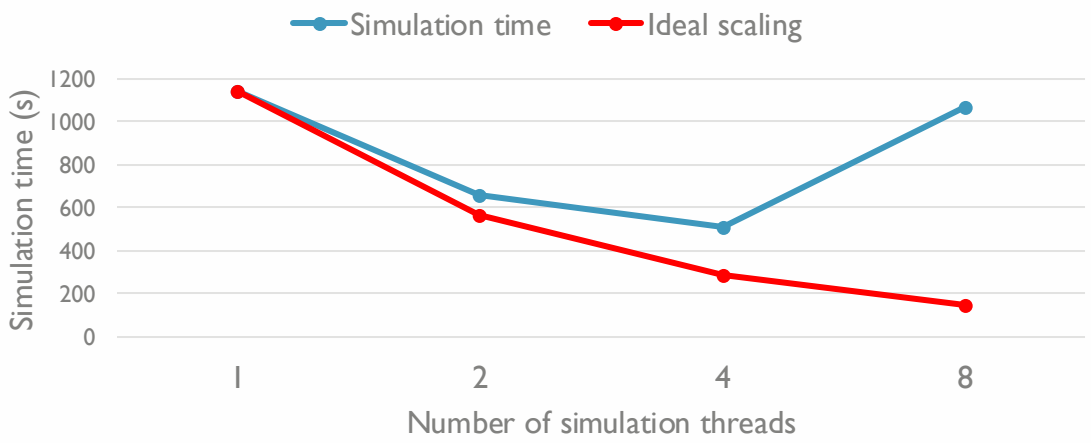
- Good scaling when the number of CPU increases with 1 thread:
  - Parallel cache subsystems
- Good scaling when the number of threads increases with 1 CPU:
  - Data shared from the same cache subsystem
- Poor scaling when the number of CPU increases with 4 threads:
  - Data shared from different cache subsystem
  - Not enough work for 16 SW threads

Demo 1: Single-node simulation with an OpenMP application

2nd experiment: simulation scaling

Number of simulation threads	1	2	4	8
Simulation time (s)	1140	655	510	1066
Million of instr. per second	0.08	0.13	0.17	0.08

Scaling of the simulation with 4 CPU, 4 threads, Seq_{len} = 64, D_{model}=128, heads = 8.



The simulation fails to scale for the same reasons as the simulated system!

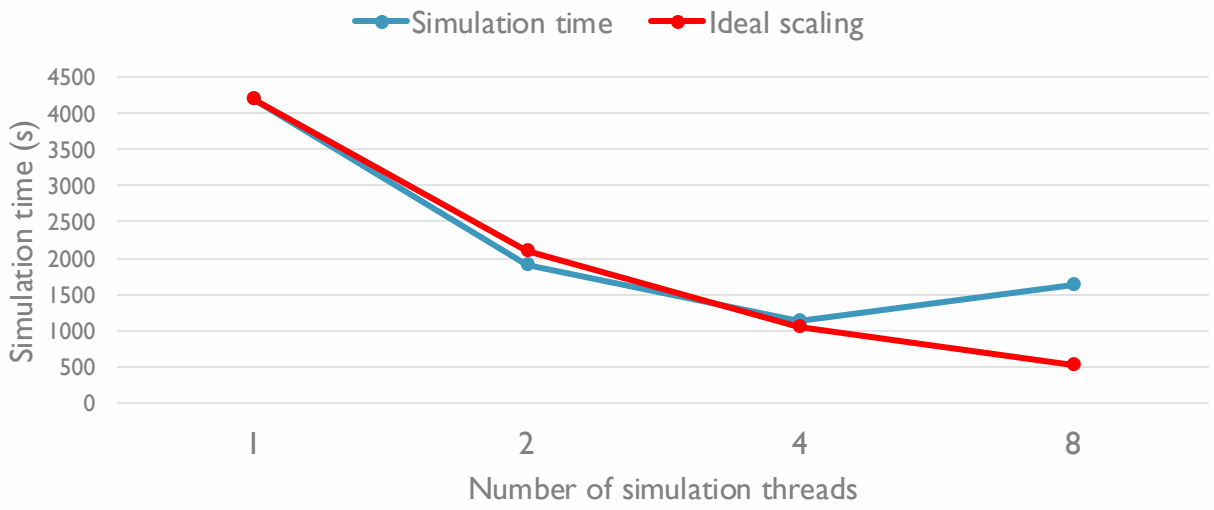
- Amdahl's Law applies for the simulation framework as well:
  - We can speed up the simulation of parallel components (CPU complex)
  - We cannot speed up the simulation of shared resources (NoC, DRAM, OS)
- Allocating too much threads unbalances the simulation

Demo 1: Single-node simulation with an OpenMP application

2nd experiment: simulation scaling

Number of simulation threads	1	2	4	8
Simulation time (s)	4196	1909	1133	1639

Scaling of the simulation with 8 CPU, 1 thread, Seq_{len} = 128, D_{model}=128, heads = 16.



Simulation scales better with more parallelism!

- More CPU but same number of threads
- Larger workload

Allocating too much threads unbalances the simulation:

- Each thread simulates a set of components
- Better to overload the simulation threads (i.e., less threads than number of simulated CPU)

Demo 2: Multi-node simulation with a hybrid OpenMP + MPI application

1st experiment: Network topologies

To get start with scale-out simulation we will experiment different inter-node network topology:

- 3 types of topology are defined in ***scale_out.py***:
  - **Simple**: instantiates ***num_node_per_router*** and interconnect them with a single router
  - **Torus:** instantiates a torus network
    - ***torus_width:*** Defines the number of links between two routers
    - ***torus_shape:*** Defines the shape of the network. Each item defines the number of routers in one dimension
  - **Fat tree:** instantiates a Fat tree network
    - ***Fattree_shape:*** defines the shape of the network.
      - Syntax is *"number of router in a level : number of port per router"*

```
fattree_shape = "1,1:2,2"
```

- You can try to run a simulation with each topology

Demo 2: Multi-node simulation with a hybrid OpenMP + MPI application

2nd experiment: simulation scaling

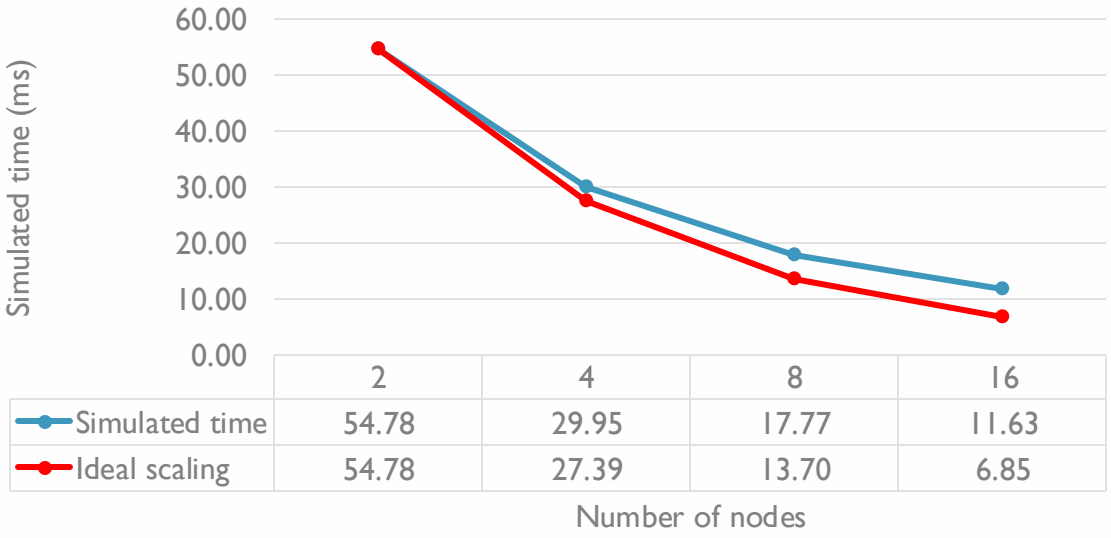
Let's explore the scalability of the **simulated system** and of the **simulation**:

- You can increase the compute power by increasing the number of nodes (e.g., 2, 4, 8). You can select any of the topologies

- You can parallelize the simulation by using multiple MPI ranks. **Make sure to not oversubscribe your computer** (i.e., no more process than number of core).
  - **mpirun --np 4 scale_out.py**

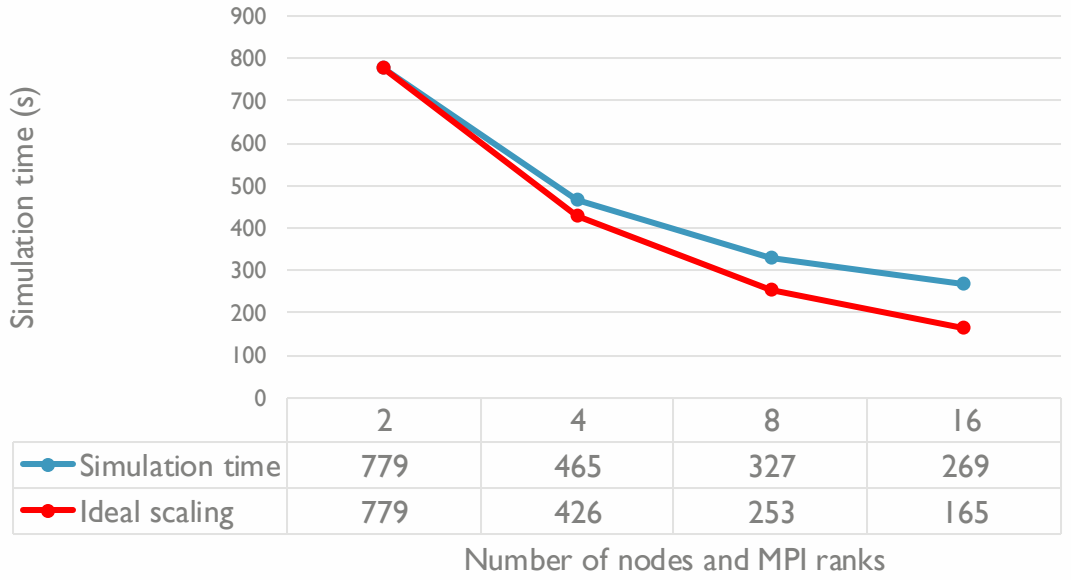
Demo 2: Multi-node simulation with a hybrid OpenMP + MPI application

2nd experiment: simulation scaling



	2	4	8	16
Simulated time	54.78	29.95	17.77	11.63
Ideal scaling	54.78	27.39	13.70	6.85

Scaling of the simulated system.
Each node integrates 1 CPU with 1 thread.
Seqlen = 128, Dmodel,=128, heads = 16.



	2	4	8	16
Simulation time	779	465	327	269
Ideal scaling	779	426	253	165

Scaling of the simulation framework.
The number of MPI ranks used matches the number of simulated node.

Ideally the simulation time would decrease with the simulated time.

Wrap-up

SST provides an open-source framework for simulating large-scale HPC systems:

- Discrete event simulation parallelized with threads and MPI ranks
- Library of computer system elements
- Python library to build scale-out system with complex network topologies
- Provides a RISC-V CPU model with an emulated operating system:
  - Allows to run multi-threaded applications
  - Allows to control a co-processor:
    - e.g., Balar:Vanadis + GPGPUSIM

SST also allows connecting "Gem5" compute core with it's uncore infrastructure.

https://www.imec-int.com/en/expertise/compute-system-architecture

Learn more about research@CSA!

https://www.imec-int.com/en/work-at-imec/job-opportunities

imec Jobs Portal

38th International Conference on VLSI Design & 24th International Conference on Embedded System

VLSI Society of India

SILICON MEETS AI : Sustainable Innovation In Accelerated Computing, Secure Connectivity, And Intelligent Mobility

4th - 8th January 2025 | Bengaluru, India