Troodon: A Unified Framework for Massively Parallel Modeling and Simulation of Computer Systems, and Digital and Analog Electronic Systems in Heterogeneous Abstraction Levels

School of Engineering

Department of Electrical Engineering

Shahed University

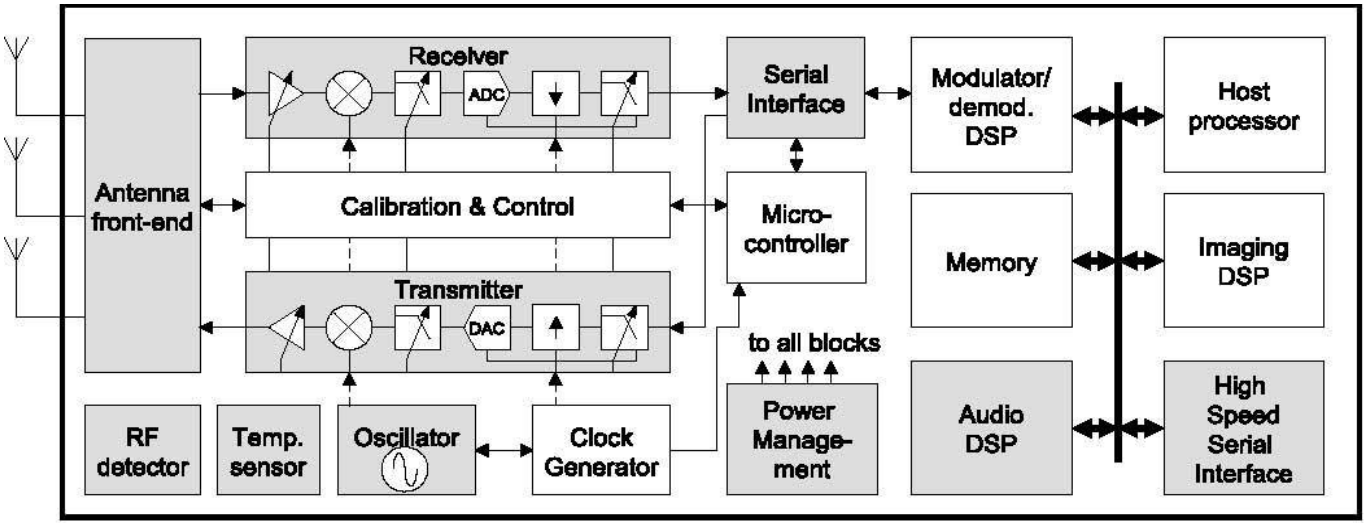
By: Alireza Poshtkohi

Website: www.poshtkohi.info

Table of Content

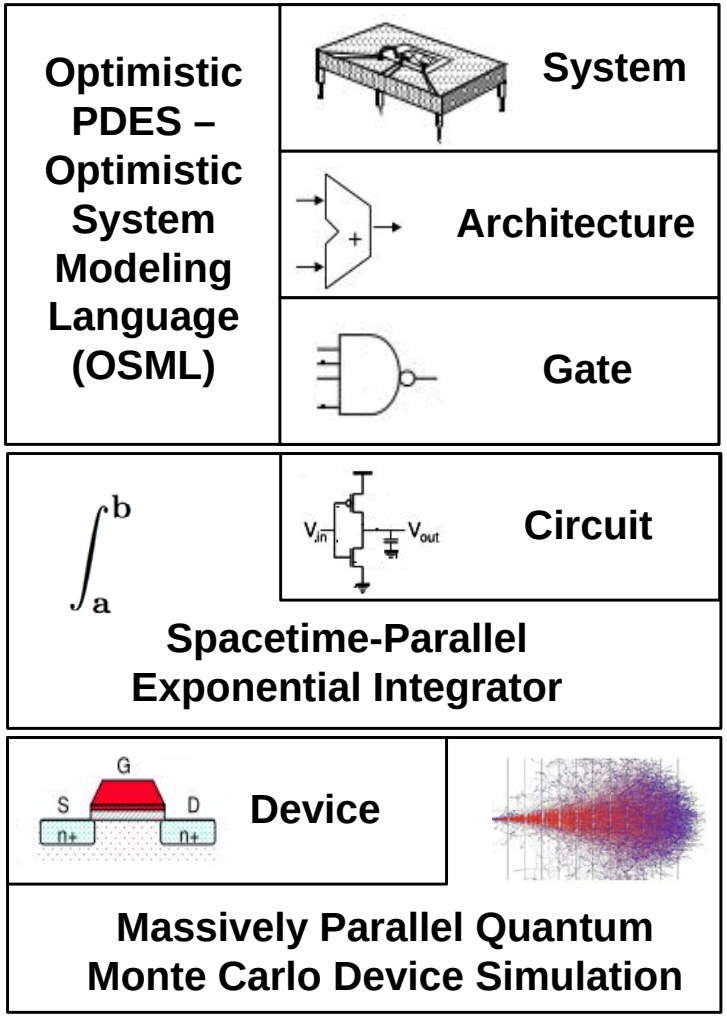
Motivation

- With increase in the complexity of integrated circuits, it is necessary to improve the speed of electronic simulators in all levels of abstraction (transistor to system level)
- We need scalable parallel modeling and simulation languages and frameworks
- Interaction between HW/SW systems and their analog physical environment
- Today systems leads to systems of digital HW/SW interwoven with analog and mixed signal blocks. Embedded Analog/Mixed-Signal (E-AMS) systems



Research Roadmap

- Parallelization approaches developed for this research



OSML Language - Introduction

- The large size and complexity of today's digital systems with heterogeneous components, and complex connections and functionalities impose many challenges for verification in different abstraction levels particularly for multi-domain systems such as analogue mixed signal.
- Accurate and high-speed simulation is a key point to enable
  - efficient and potential verification
  - power and performance estimation,
  - and design space exploration.
- These simulations are performed based on discrete event simulation.

OSML Language - Contributions

- A new parallel simulation language (PSL) for ESL
  - A new optimistic PDES language called Optimistic System Modeling Language (OSML) for ESL
  - One of the six research challenges in PADS
  - Solves the obstacles unexplored by traditional PSLs
  - Help ESL community avoid repetition of basic discoveries
  - First application of optimistic PDES to SLDLs
  - For the first time, we allow different hardware models at different electronic abstraction levels to be executed by optimistic synchronization
  - We implement a unified ESL CAD tool called Troodon to automate parallelization of the existing hardware languages

Parallel Simulation - Principles

- Execution of a discrete event simulation on a parallel or distributed system with several physical processors.
- The simulation model is decomposed into several sub-models that can be executed in parallel
  - Spatial partitioning,
  - Temporal partitioning,
- Radically different from simple simulation replications.

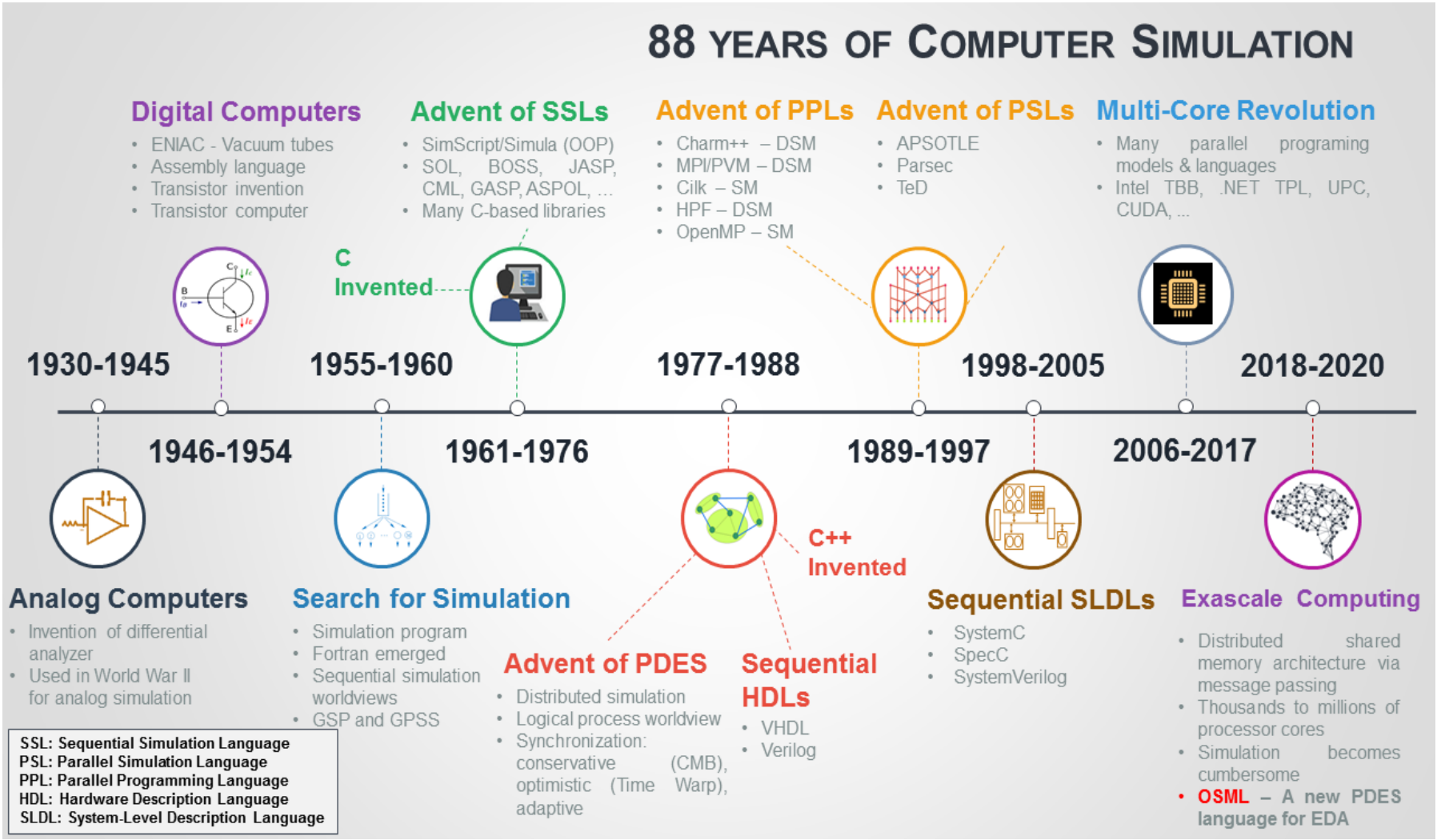
Synchronization Protocols

- Fundamental concepts
  - Each Logical Process (LP) can be at a different simulation time.
  - Local causality constraint: events in each LP must be executed in time stamp order.
- Synchronization algorithms
  - Conservative: avoids local causality violations by waiting until it's safe.
  - Optimistic: allows local causality violations but provisions are done to recover from them at runtime.

Time Warp Algorithm (Jefferson)

- Different rollback management techniques
- Copy State Saving (CSS)
  - Efficient if LP state small
  - Can be made transparent to application
- Periodic State Saving (SS)
  - Must turn off message sending during coast forward
  - Reduced memory requirements
  - less time for state saving
  - Increased rollback cost
- Incremental State Saving (ISS)
  - Preferred approach if large state vectors
  - Means to simplify usage required
- Reverse Computation (RC)
  - Efficient, requires automation

OSML Language – Related Work



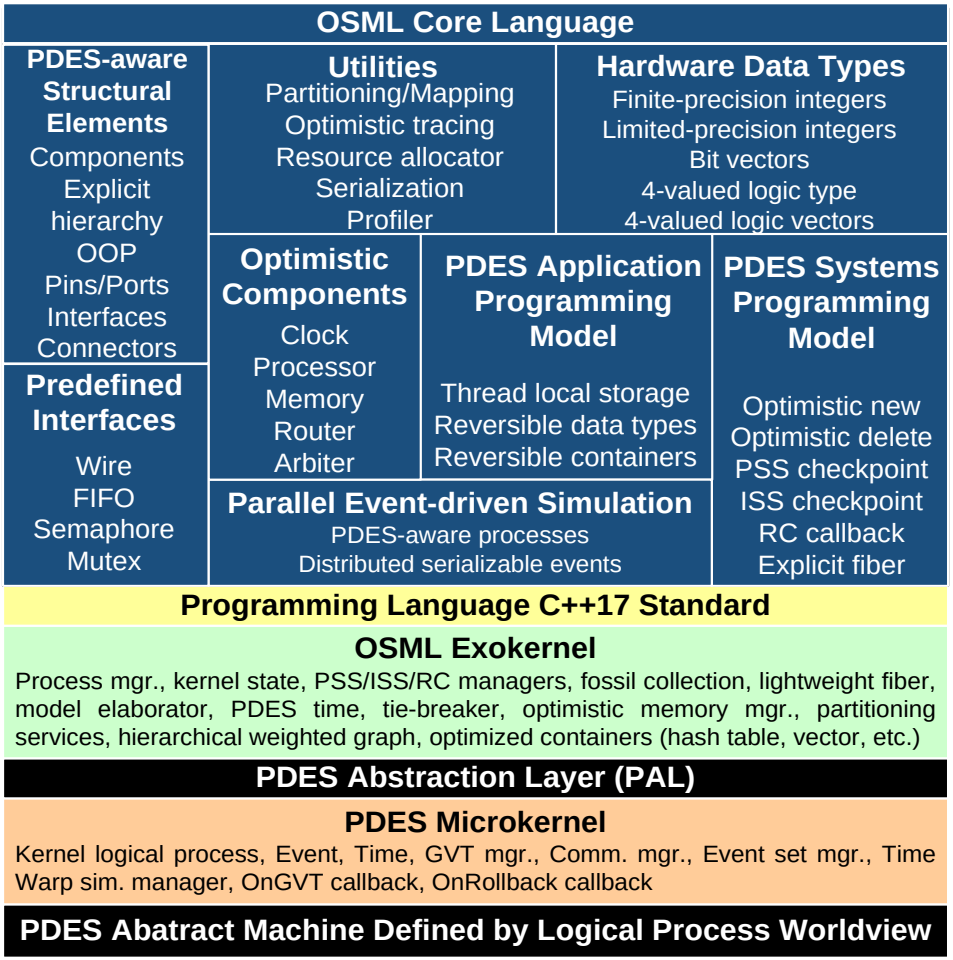
A. Poshtkohi, M.B. Ghaznavi-Ghoushchi, K. Saghafi, *Computer Modeling and Simulation – A Philosophical and Technological Survey*, under finalization, 2019.

OSML Language

- OSML is a fully-fledged object-oriented simulation language
  - Process-level granularity
  - General purpose and domain-specific PSL
  - with a dedicated elaboration phase
  - has a PDES-aware syntax
  - is a *PDES systems programming language* for explicit hybrid state management
  - is a *PDES application programming language* for transparent hybrid state management
  - takes advantage of a profile-driven sequential execution for partitioning
  - is a general-purpose and special-purpose simulation language
  - allow the programmer to use the facilities present in common parallel programming languages like MPI (mapping, partition, etc.)
  - Reversible ADT
  - Crafted in C++17
  - and so many other features

OSML Language

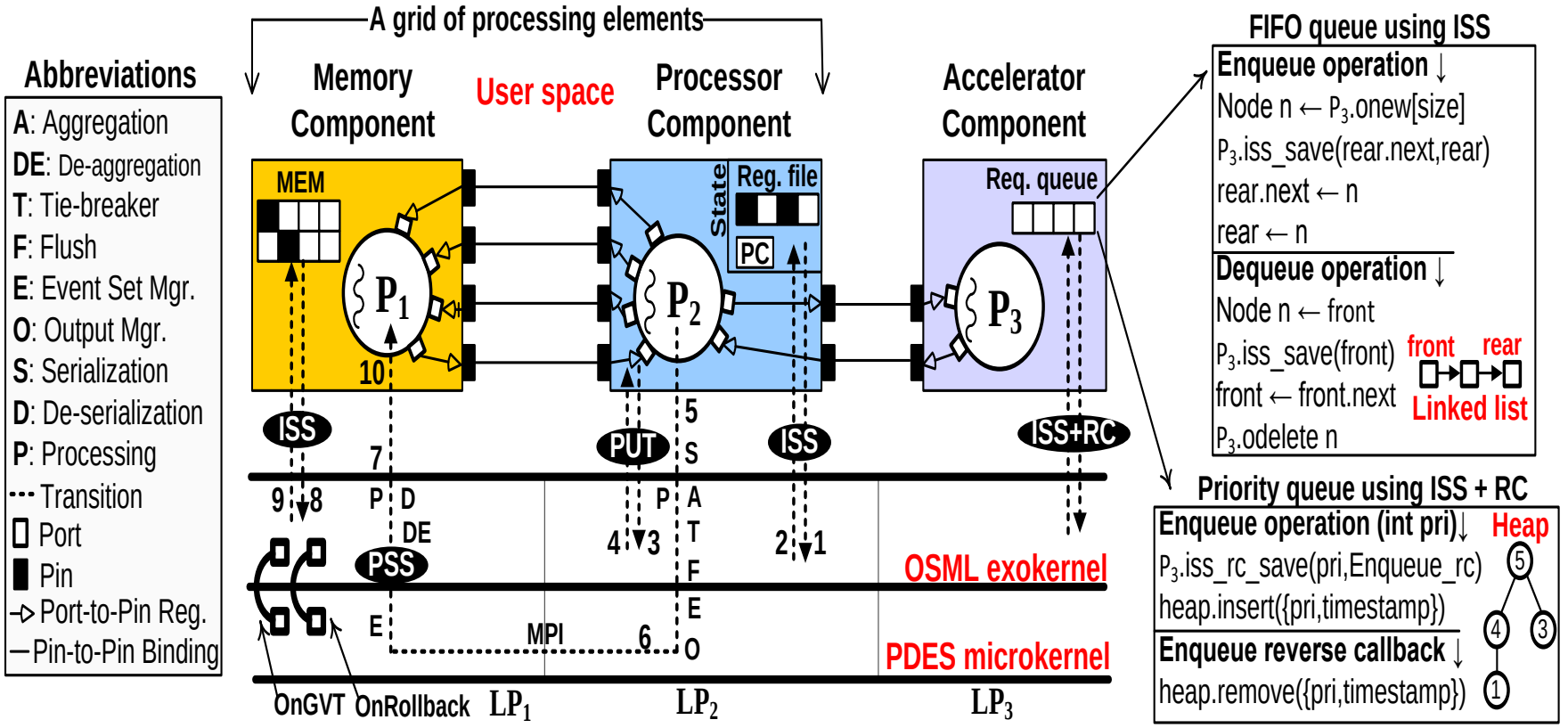
- One of the primary goals of OSML language is to enable optimistic, explicitly parallel system-level modeling



OSML language architecture

OSML's PDES Systems Programming Model

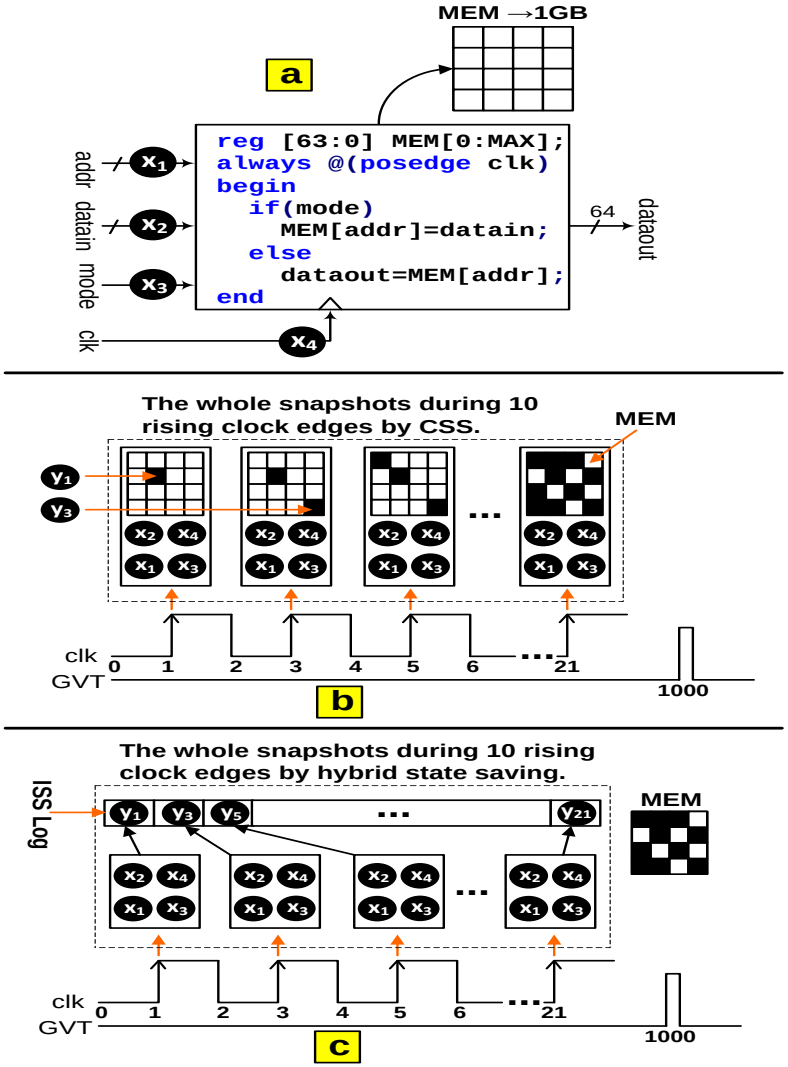
- It provides a user with low-level access to optimistic dynamic memory and hybrid state saving for developing hardware models



A many-core architecture of PEs and its interaction with optimistic PDES engine by hybrid checkpointing

OSML's PDES Systems Programming Model

- It provides a user with low-level access to optimistic dynamic memory and hybrid state saving for developing hardware models



OSML's PDES Systems Programming Model

- Molding an optimistic component in OSML

```
1 template <class T, int size>
2 class osml_sync_memory : public osml_component {
    // Pin and port definitions
3   osml_pin<osml_wire<bool > > clk; ...
4   osml_inport<osml_wire<bool > > _clk;…
5   process_state pss_state; // PSS-based state definition
6   T *array = nullptr; osml_process *p;
7   osml_sync_memory(const std::string &name){
8     register_pin(clk, _clk); ... // Register pins to ports
9     // Register a stackful process and do its settings
10    p = register_process(process, true, &pss_state, name);
11    p->register_on_partitioning(on_partitioning);
12    p->register_port(_clk); ...
13    p->register_sensitivity(_clk, OSML_POS_EDGE);
14  }
    // This method is invoked after partitioning
15  static void on_partitioning(osml_process &owner){
16    auto myComp = (osml_sync_memory *)owner.get_component();
17    myComp->array = new T[size];
18  }
};
```

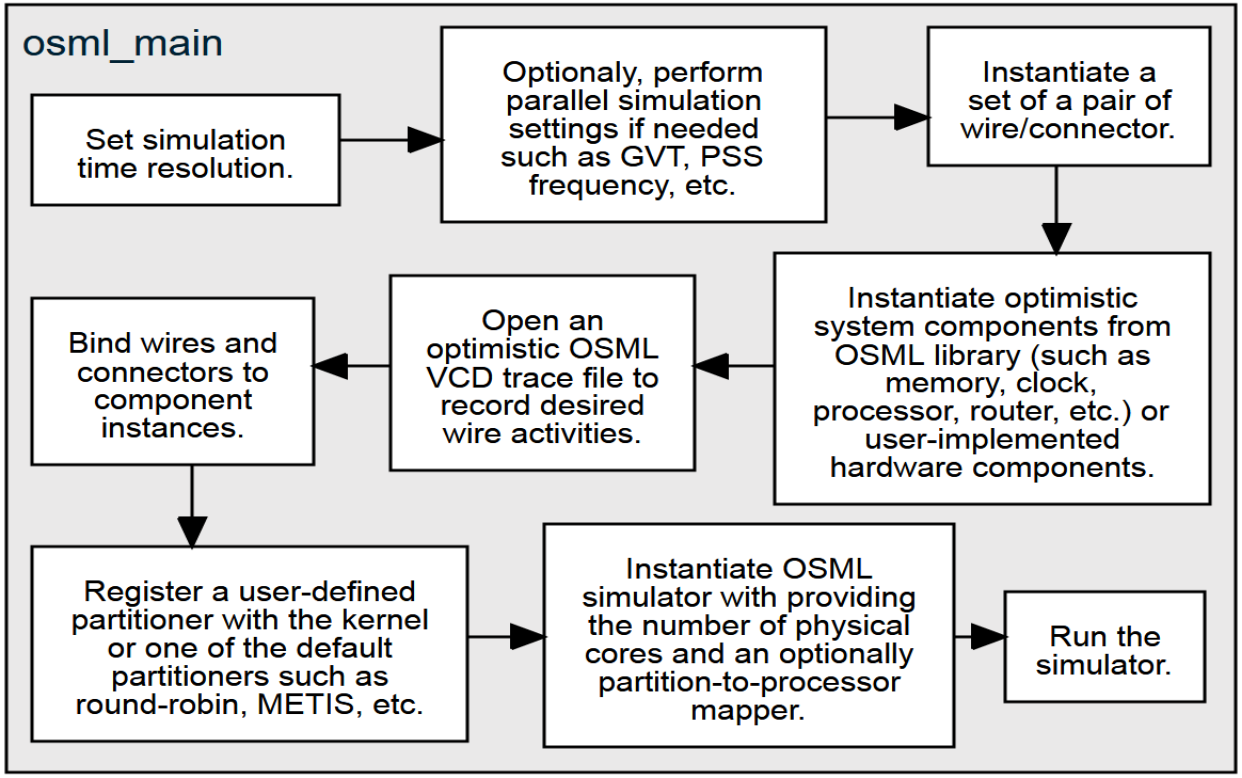
Definition of an optimistic memory written in OSML

```
1 static void process(osml_process &owner){
2   auto myComp = (osml_sync_memory *)owner.get_component();
3   auto myState = (process_state *)owner.get_state();
4   int addr = myComp->address->fetch();
5   switch(myState->get_label()) { case 0: goto L0; }
6   while(true) {
7     if(!myComp->_we->fetch() …) // Read from memory
8       myComp->_data_out->put(myComp->array[addr], owner);
9     else if(myComp->_we->fetch() …) { // Write to memory
        // We must checkpoint before performing the real write
10      owner.iss_save(&myComp->array[addr], sizeof(T));
        // Now, we perform the write to the memory
11      myComp->array[addr] = myComp->data_in->fetch();
12      myState->writes++;
13    }
14    osml_wait(owner); myState->set_label(0);return;L0:
15  }
}
```

The process implementation of the memory component

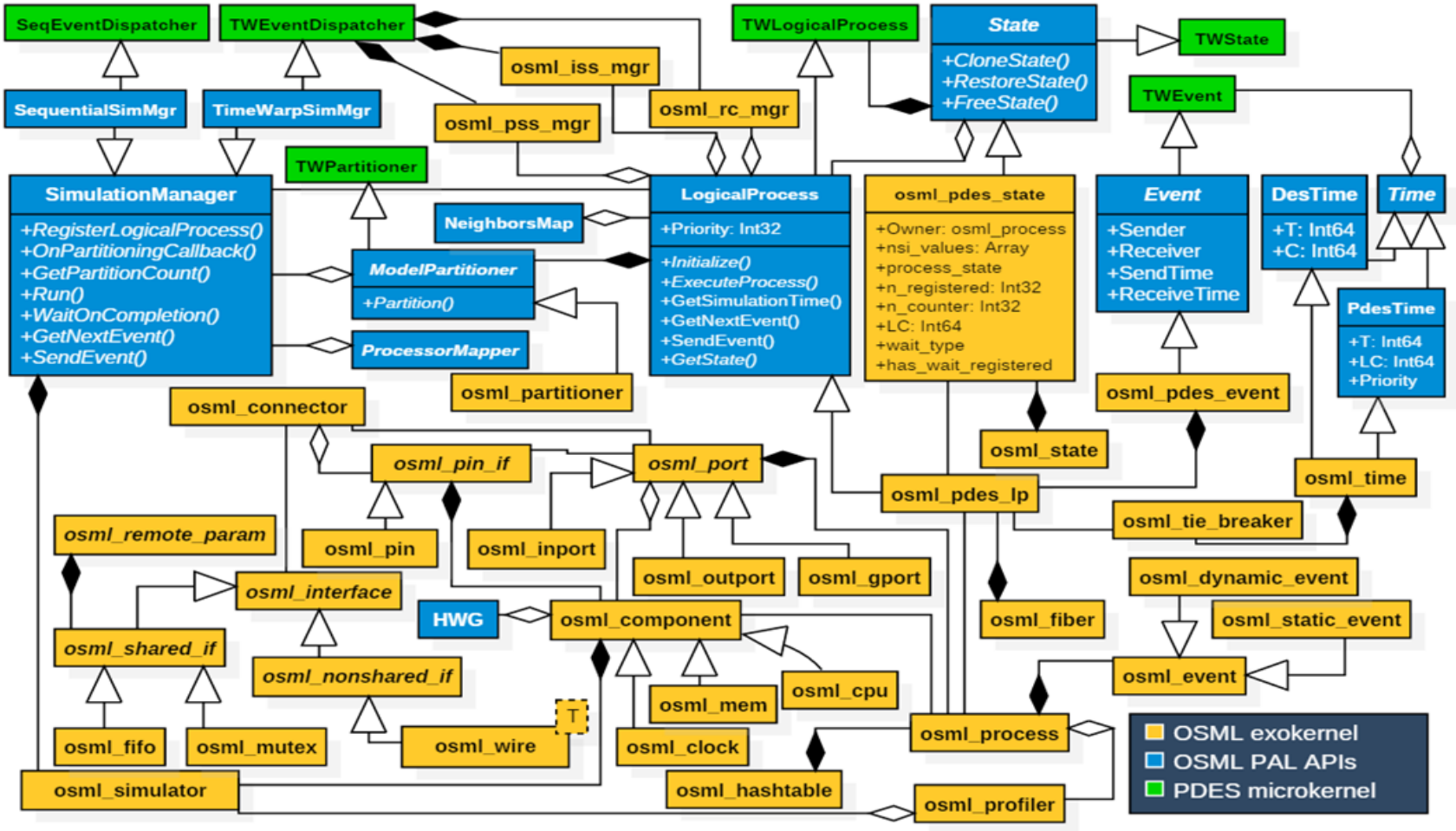
OSML's PDES Systems Programming Model

- Following the development of optimistic components, the programmer connects them together in a function called *osml_main*



The Optimistic OSML Simulation Kernel

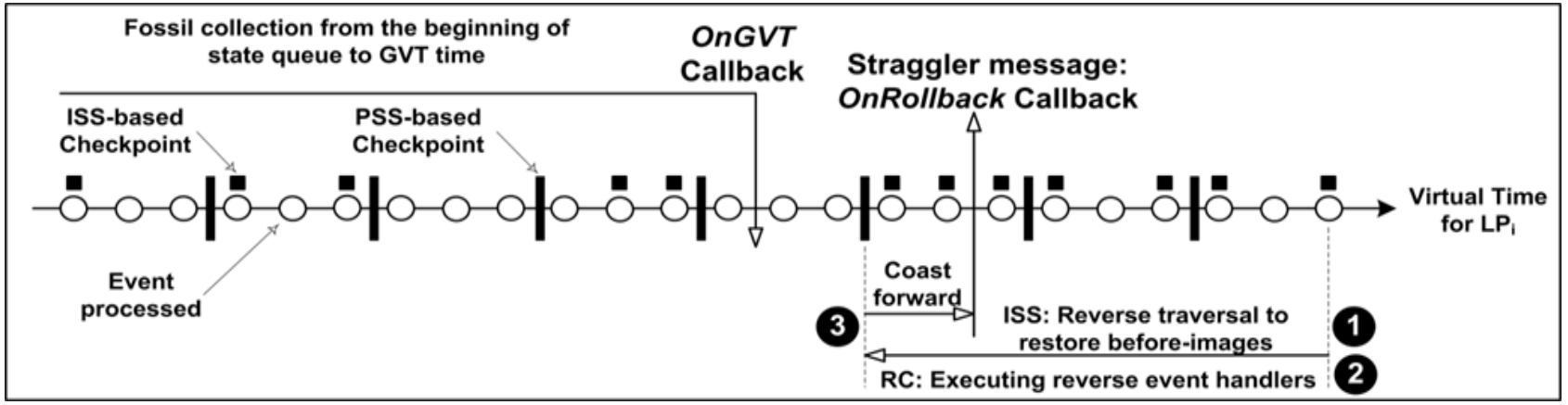
- The system-level description written in OSML language must finally be simulated by a kernel



UML class diagram of the optimistic OSML simulation kernel

The Optimistic OSML Simulation Kernel

- When a straggler message is received, rollback is performed in three mixed steps
  - ISS manager restores the before-images of state variables by a reverse traversal
  - RC manager invokes the reverse event handlers upon completion of every ISS operation to backtrack the computation
  - PSS manager is activated and performs coast forward in which the LP is executed up to the rollback point to replay intermediate process states



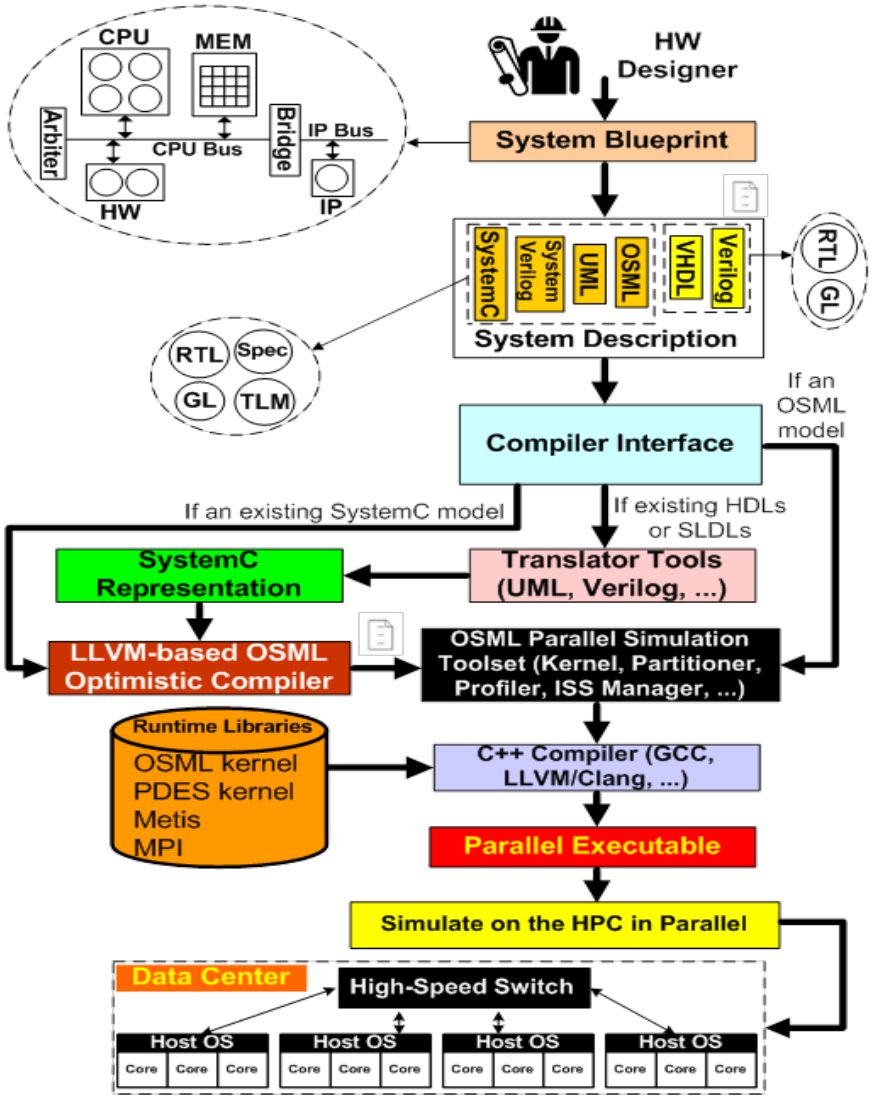
The Optimistic OSML Simulation Kernel

- The problem of simultaneous events in parallel simulation
  - Two or more events that are scheduled to occur at the same point in time are called simultaneous events
  - A number of tie-breaking rules are necessary to decide which simultaneous event should be executed first for a deterministic simulation
  - To solve this problem, time is defined by the following rule

$$t_1 < t_2 \Leftrightarrow T_1 < T_2 \vee (T_1 = T_2 \wedge LC_1 < LC_2) \vee (T_1 = T_2 \wedge LC_1 = LC_2 \wedge Pr_1 < Pr_2)$$

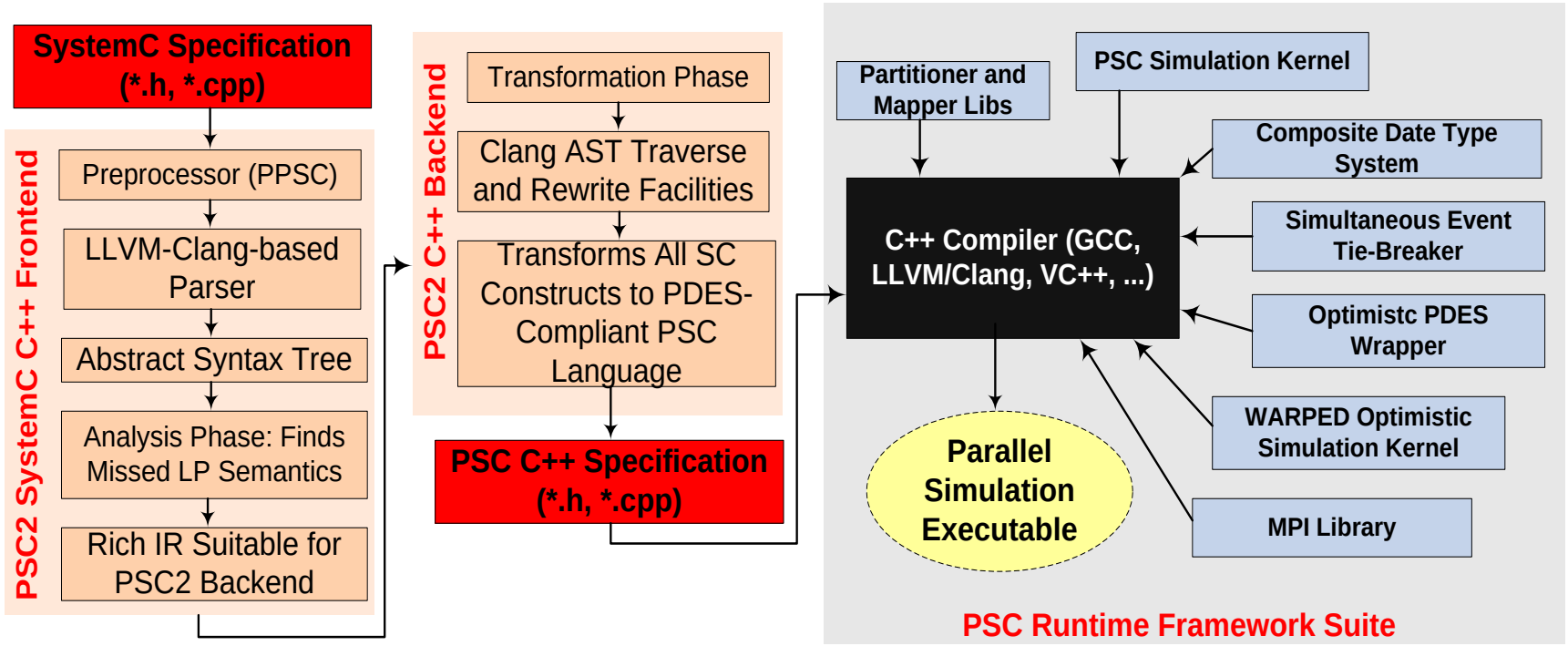
Troodon Tool Flow

- Troodon facilities the task of parallel simulation of existing HDLs and SLDLs on many-core HPC clusters



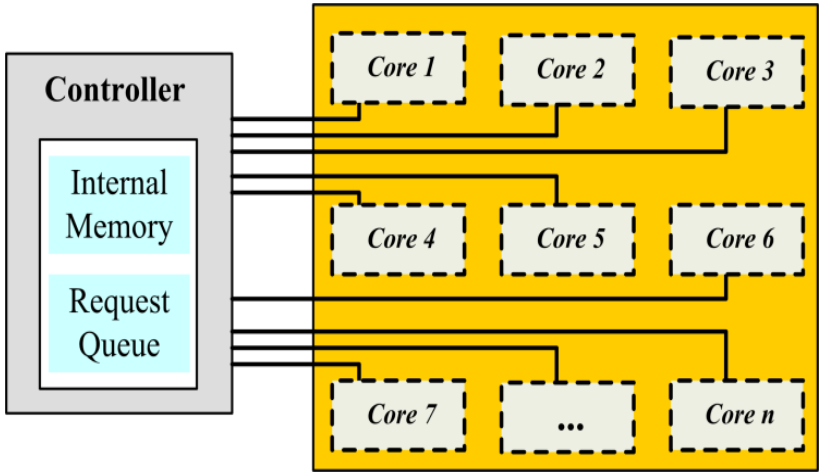
Troodon Tool Flow

- SystemC-to-OSML compiler works in three phases:
  - Preprocessing Phase
  - Compiler Analysis Phase (Frontend)
  - Source-to-Source Transformation Phase (Backend)

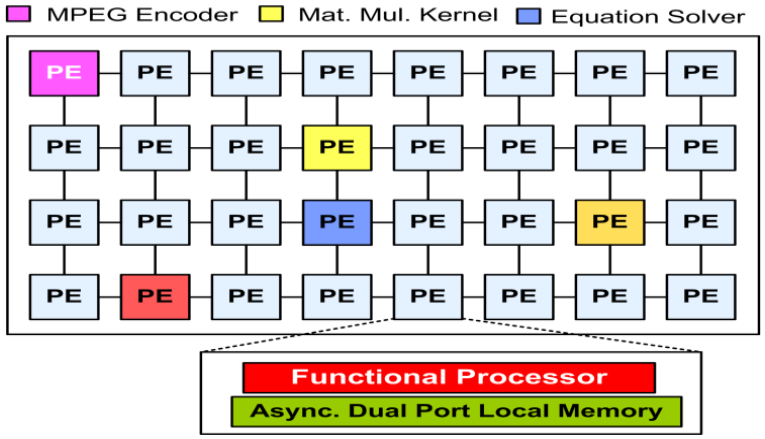


OSML Language – Performance Study

- We study a number of experiments to evaluate the OSML performance
  - The tests were carried out on an HPC cluster of 17 nodes
  - Each machine had 12 cores operating at 3.33 GHz, 12GB RAM and 12MB L3 cache
  - The machines were connected to a 40 GB/s fiber-optic network
  - Each node ran CentOS 7 Linux with a kernel 3.10.0-327
  - OSML speedups are reported in comparison with Accellera sequential SystemC reference simulator



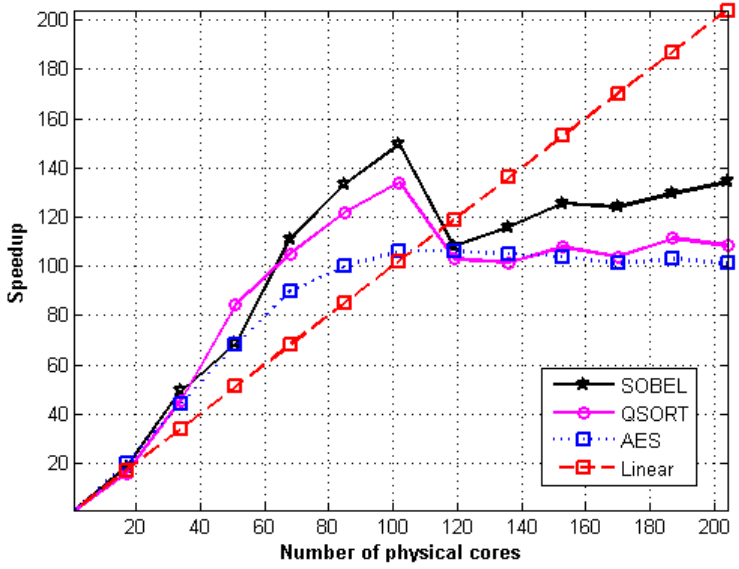
A many-core accelerator



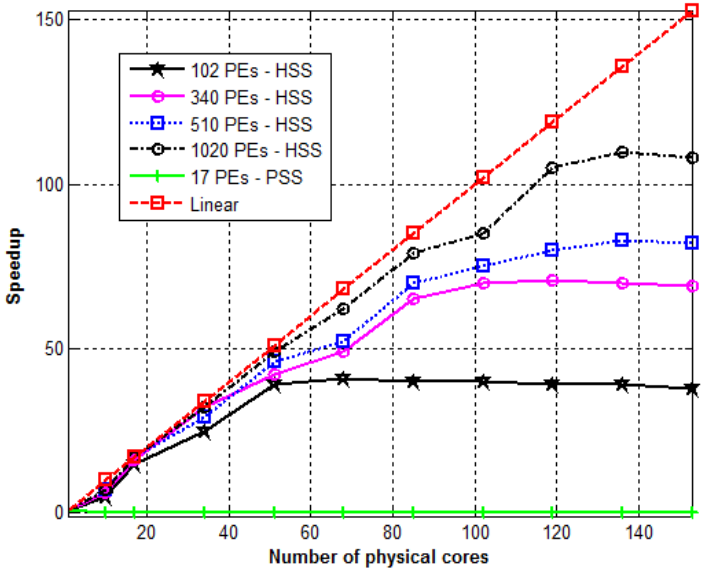
A many-core BFM platform

OSML Language – Performance Study

- Speedups of the benchmarks



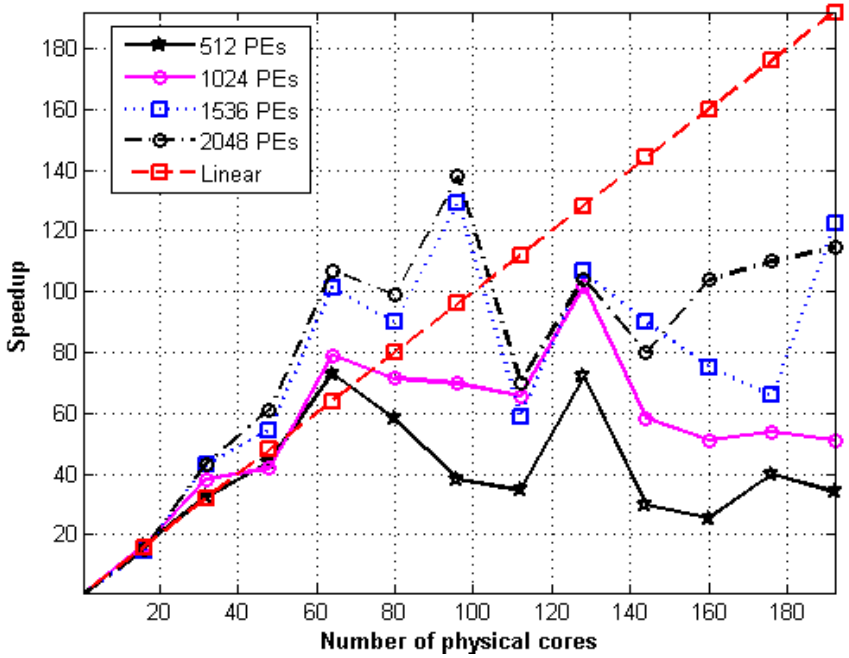
Speedups of the different system level models



Speedups of the BFM model with respect to the number of PEs

OSML Language – Performance Study

- Speedups of the benchmarks
  - This model has fluctuations in speedups specially by increasing the number of processors. This is because of the small amount of RAM installed per node on the cluster



Speedups of the many-core triple DES RTL model with respect to the number of PEs.

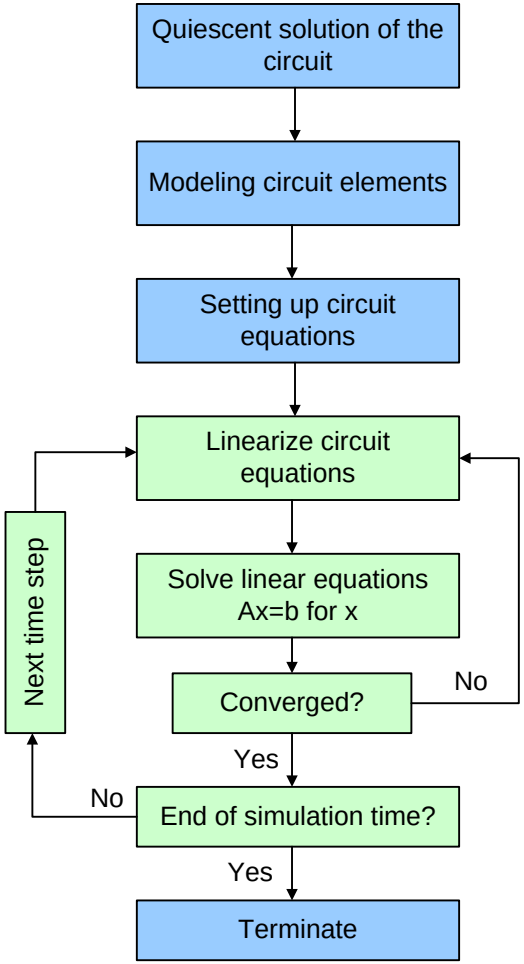
A. Poshtkohi, M.B. Ghaznavi-Ghoushchi, K. Saghafi, *Optimistic Modeling and Simulation of Complex Hardware Platforms and Embedded Systems on Many-Core HPC Clusters*, IEEE Transactions on Parallel and Distributed Systems, 30:2 (2019), 428-444. (IF=3.971), doi: 10.1109/TPDS.2018.2860014.
A. Poshtkohi, M.B. Ghaznavi-Ghoushchi, K. Saghafi, *Optimistic Synchronization and Tool Flow for Massively Accelerating SystemC on Many-Core HPC Clusters*, 2019. In finalization.

Spacetime-Parallel Integration Using Exponential Integrators

- Exponential integrators are a class of numerical methods for the solution of ordinary differential equations, specifically initial value problems
- This large class of methods from numerical analysis is based on the exact integration of the linear part of the initial value problem.
- Because the linear part is integrated exactly, this can help to mitigate the stiffness of a differential equation.

Spacetime-Parallel Integration Using Exponential Integrators

- Related works
  - Traditional SPICE simulation: XYCE parallel simulator [1]



[1] Online: https://xyce.sandia.gov/

Spacetime-Parallel Integration Using Exponential Integrators

- Sequential formulation

$$\frac{du(t)}{dt} = Au(t) + g(t)$$

$$e^{-At}\frac{du(t)}{dt} = e^{-At}Au(t) + e^{-At}g(t)$$

$$\frac{d}{dt}(e^{-At}u(t)) = -Ae^{-At}u(t) + e^{-At}\frac{du(t)}{dt}$$

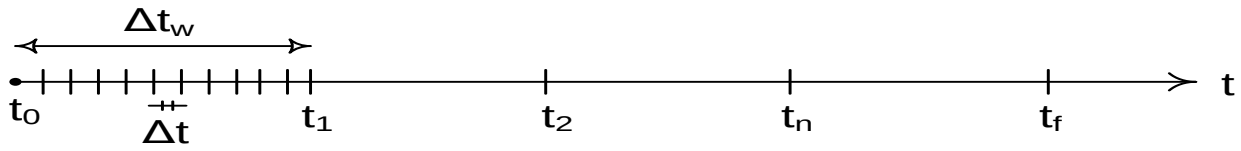
$$\frac{d}{dt}(e^{-At}u) = e^{-At}g(t)$$

$$e^{-At}u(t) - e^{-At_0}u(t_0) = \int_{t_0}^{t} e^{-A\tau}g(\tau)d\tau$$

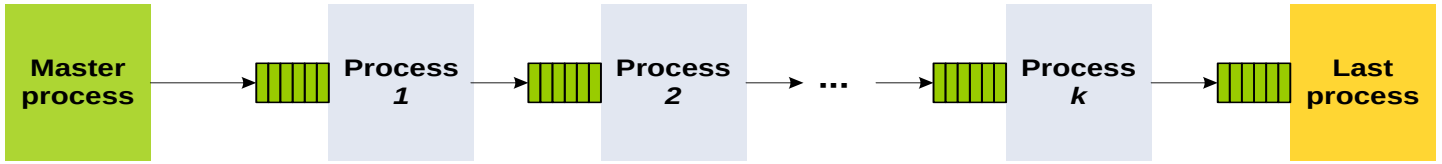
$$u(t) = e^{A(t-t_0)}u(t_0) + \int_{t_0}^{t} e^{A(t-\tau)}g(\tau)d\tau$$

Spacetime-Parallel Integration Using Exponential Integrators

- Parallel formulation



Scheduling of Moving Exponential Time Windows (METW) technique



A pipeline of processors in METW

Spacetime-Parallel Integration Using Exponential Integrators

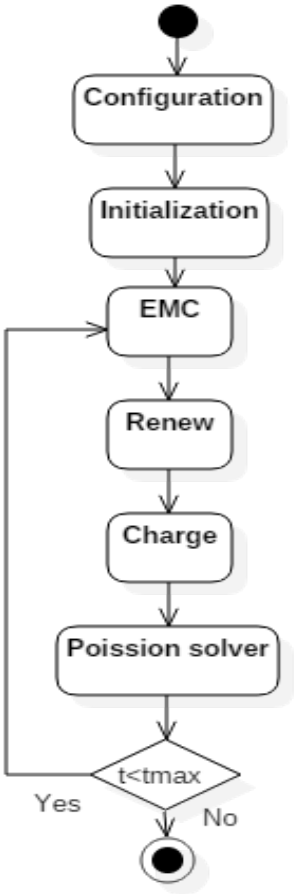
- Experimental studies
  - Implemented in C++ and MPI
  - Tested on a 12-core machine (NUMA, two processors each having 6 cores)
  - A RLC network with 100k elements
  - Spacetime simulator is only implemented to exploit time-axis parallelism not space
  - Xyce is a space-parallel SPICE simulator
  - Xyce is set with preconditioned GMRES iterative solver
  - The results shows promising results that can be improved if implemented by space parallelization

Simulator	Execution times (sec)	Parallel execution times (sec)			
		6 cores		12 cores	
		Time	Speedup	Time	Speedup
Spacetime	110.26	59.1	1.86	29.7	3.71
Xyce	2300.78	1114.34	2.06	590.53	3.89

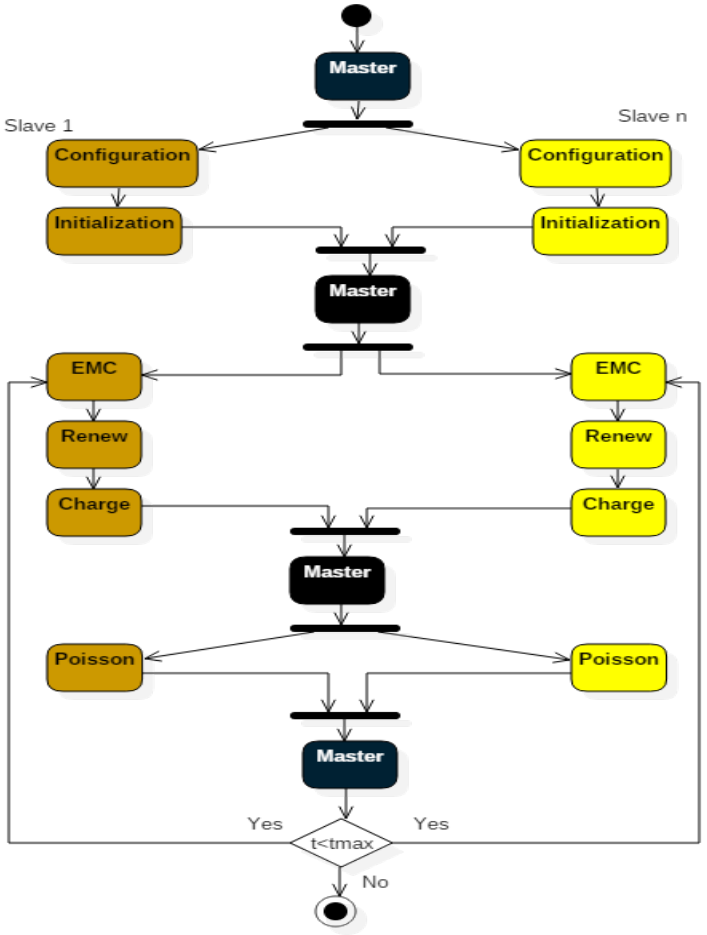
A. Poshtkohi, M.B. Ghaznavi-Ghoushchi, K. Saghafi, *Extreme-Scale Spacetime-Parallel Modeling and Simulation of Ordinary and Partial Differential Equations using Exponential Integrators*, under finalization, 2019.

Parallelization Techniques Not Discussed in this Presentation

- Parallel Nano-Scale Device Level Monte-carlo Simulation
  - Device-level simulation are very time-consuming, so it requires parallelization



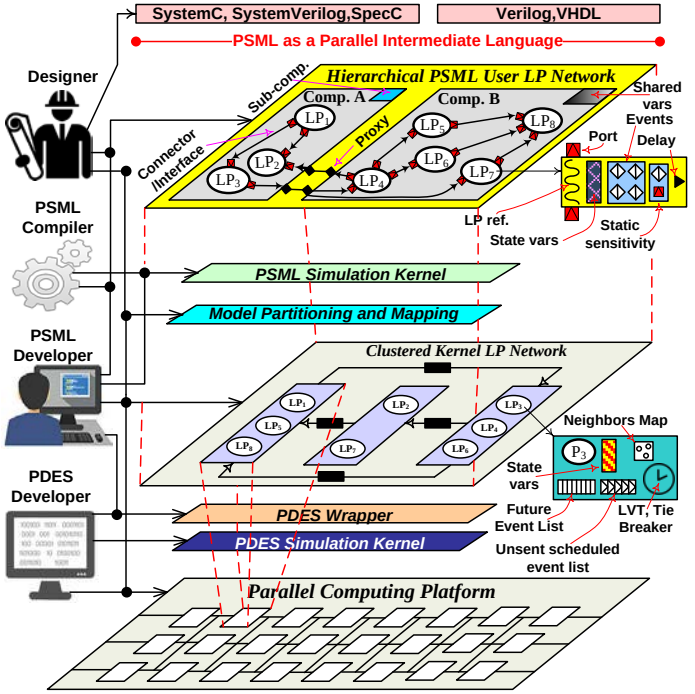
Sequential Monte-carlo Device Simulation



Direct parallelization of the monte-carlo device simulations

Parallelization Techniques Not Discussed in this Presentation

- PSML: Parallel System Modeling and Simulation Language for Electronic System Level
  - An asynchronous conservative language
  - Cross-platform execution environment using Parvicursor services
  - Formally-defined parallel execution semantics
  - A multi-core implementation

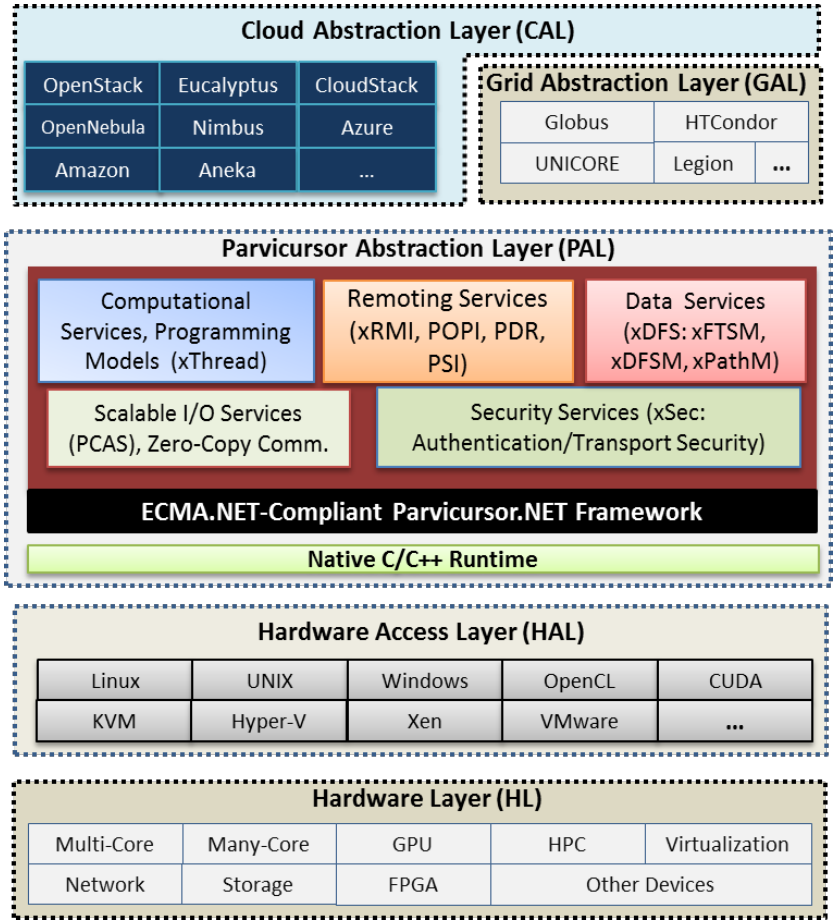


PSML conceptual framework

A. Poshtkohi, M.B. Ghaznavi-Ghoushchi, K. Saghafi, *PSML: Parallel System Modeling and Simulation Language for Electronic System Level*, Journal of Supercomputing, 2018. doi: 10.1007/s11227-018-2682-1

Parallelization Techniques Not Discussed in this Presentation

- Parvicursor is based on the distributed objects paradigm that can facilitate the construction of scalable and high-performance parallel distributed systems



A. Poshtkohi, M. B. Ghaznavi-Ghoushchi, K. Saghafi, *The Parvicursor Infrastructure to Facilitate the Design of Grid and Cloud Computing Systems*, Computing, vol. 99, pp. 979–1006, October 2017.

Contributions

- **Parvicursor Infrastructure**
- **PSML Language**
  - A conservative PDES language
  - Mathematically formalization of its parallel execution semantics
  - PSML simulation kernel and multi-core optimizations
- **OSML Language**
  - For the first time, application of optimistic PDES to SLDLs in ESL
  - For the first time, execution of different hardware models at different electronic abstraction levels using optimistic synchronization by proposing a hybrid state saving scheme
  - A new optimistic PDES language called *Optimistic System Modeling Language (OSML)* along with its distributed simulation kernel

Contributions

- **Troodon CAD Tool**
  - Automatic parallelization of HDLs and SLDLs
  - Parallel language extensions to the SystemC standard
  - SystemC-to-OSML compiler infrastructure
  - A new timing model for SystemC-compliant OSML execution
- **Spacetime-Parallel Exponential Integrator**
  - Developing a new approach for spacetime parallel simulation of SPICE-like simulators
  - Parallel integrations methods
  - Discretization techniques for exponential integrators
  - Preconditioner for exponential integrators
  - A new approach for computing the action of matrix exponential upon a vector
  - Formulation of electrical circuits for exponential integrators
- **Parallel Device-Level Simulation**

Future Directions

- **Development of Parvicursor Services for Cloud Computing**
  - such as sandboxing, more performant event-driven file transfer, etc.
- **OSML Language**
  - Applications to other domains such as computer networks
  - Dynamic load-balancing and adaptive PDES protocols
  - Optimizing simulation speed using deep learning
  - Seamless interfacing with other non-simulation parallel programming languages
- **Parallel Exponential Integration**
  - Full implementation of spacetime-parallel simulator for linear and non-linear circuits
  - OSML-AMS : AMS extensions to OSML language
  - Multiphysics language extensions to OSML language
  - Geometric integration
  - Domain decomposition

Thank you!