



Toolchain For Programming, Simulating and Studying the XMT Many-Core Architecture

Fuat Keceli, Alexandros Tzannes,
George C. Caragea, Rajeev Barua and Uzi Vishkin

University of Maryland, College Park, MD

XMT: eXplicit Multi-Threading (*Not Cray XMT*)

Available at:

<http://www.umiacs.umd.edu/users/vishkin/XMT/index.shtml#sw-release>

Or search **XMTC** on **SourceForge**

Overview

- XMT framework and architecture
- XMTSim
 - The cycle-accurate simulator of the architecture
- XMT Compiler
 - Optimizing compiler for explicit parallelism
- Importance of the toolchain
- Summary

Explicit Multi-Threading (XMT)

General Purpose Many-Core Computer

Framework and Architecture

Searching for parallel platforms which provide ...

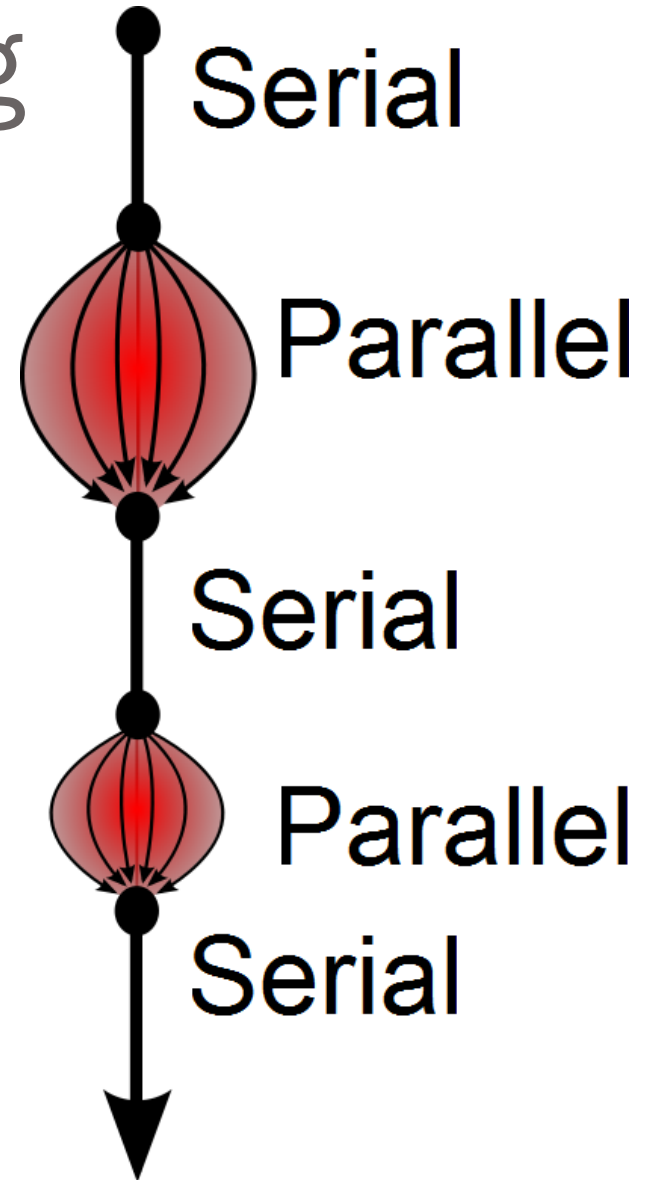
- Ease-of-programming
- Competitive performance for both high and low degrees of parallelism
- Manageable power

Explicit Multi-Threading

- “Explicitly” (definition)
 - Parallelism is explicitly exposed by the programmer
 - HP 4th ed. “If you want your program to run significantly faster, you're going to have to parallelize it”

XMTC Programming

Execution Modes:
Serial and parallel



XMTC Programming

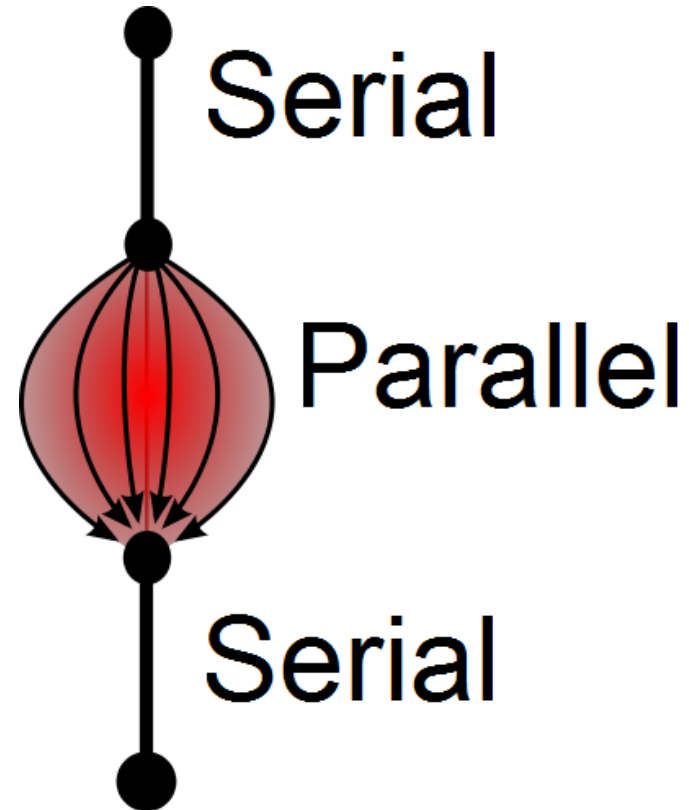
```
start  
serial code  
...
```

```
spawn(# threads) {  
    parallel SPMD code*  
}
```

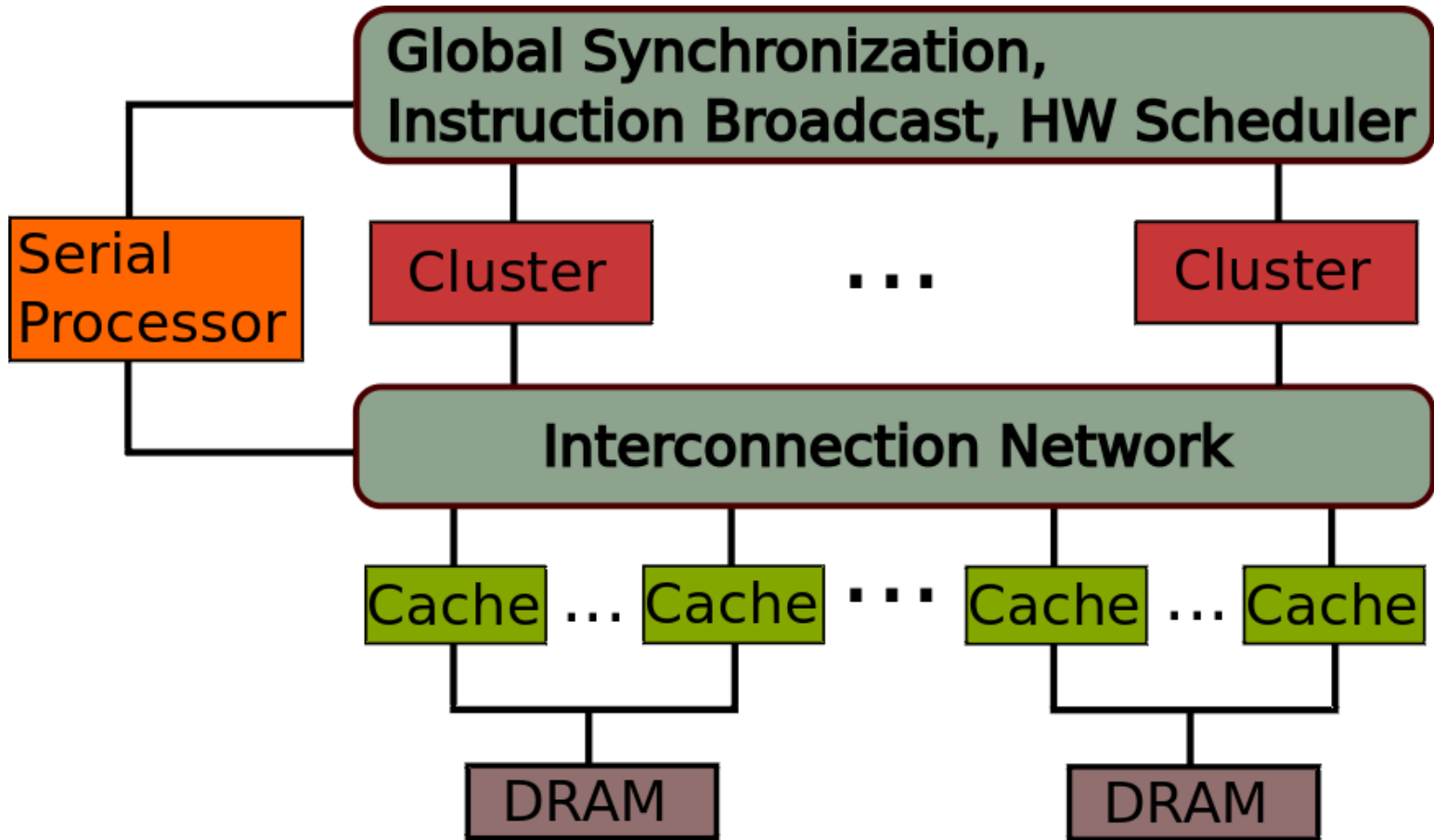
```
serial code  
...
```

```
halt
```

* *Not MPI type SPMD.*

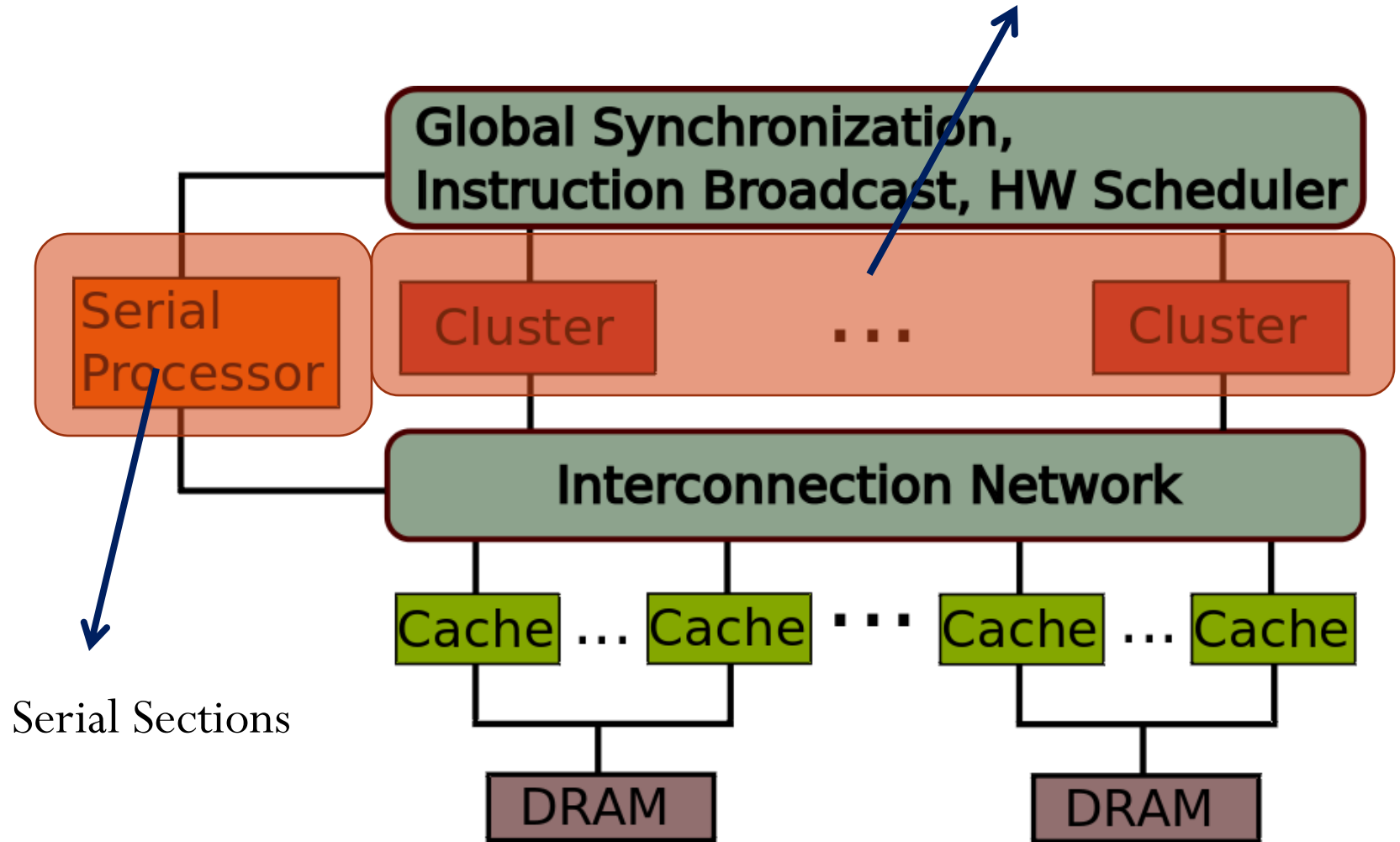


XMT Architecture

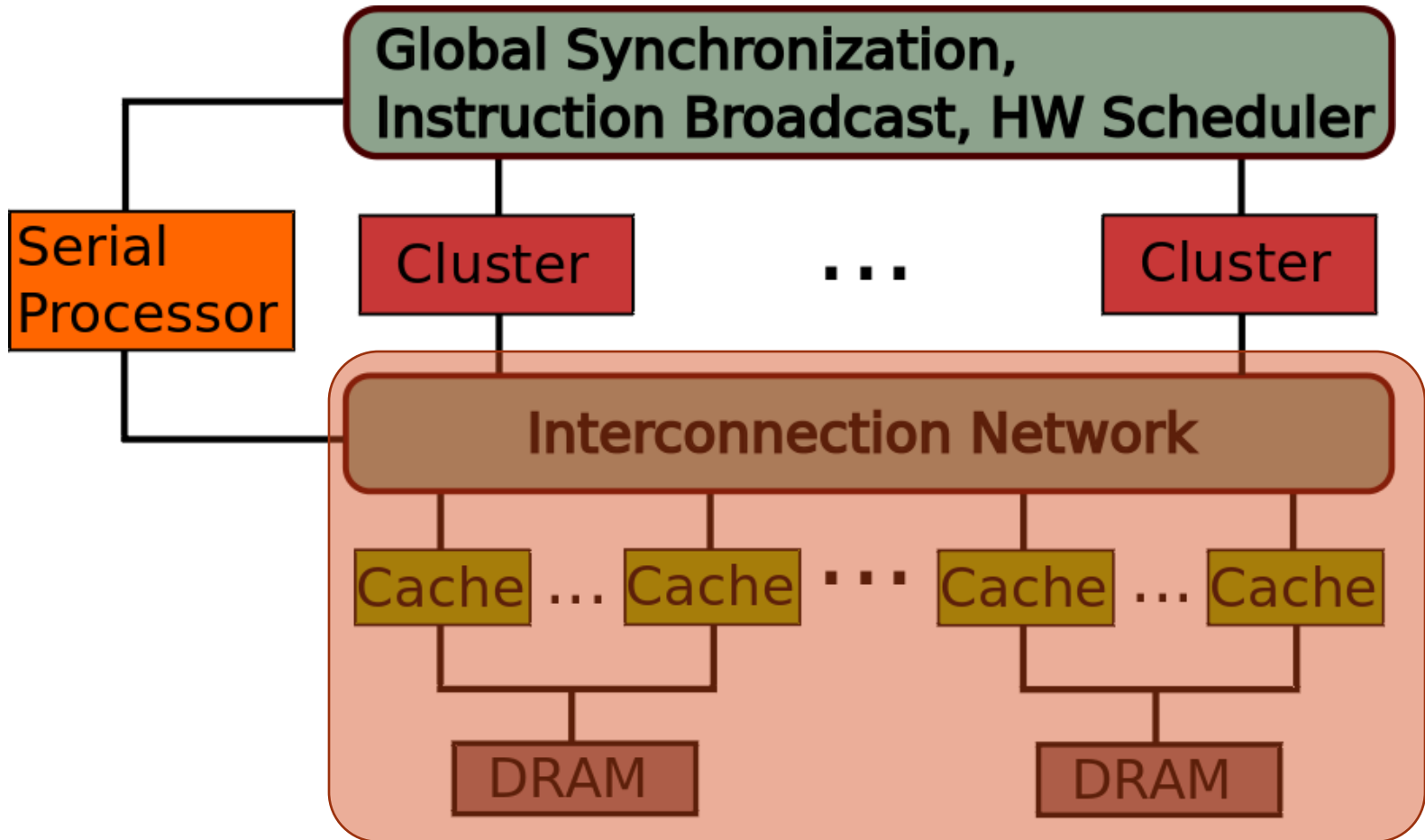


XMT Architecture

Parallel sections



XMT Architecture



Explicit Multi-Threading

- “Explicitly” (definition)
 - Parallelism is explicitly exposed by the programmer
 - HP 4th ed. “If you want your program to run significantly faster, you're going to have to parallelize it”
- Why focus on Explicitness
 - Automatic parallelization: limited scale & generality
 - Irregular parallelism
 - Serial coding sometimes hides parallelism (BFS).

A Holistic Approach

Vertical Stack of Programming

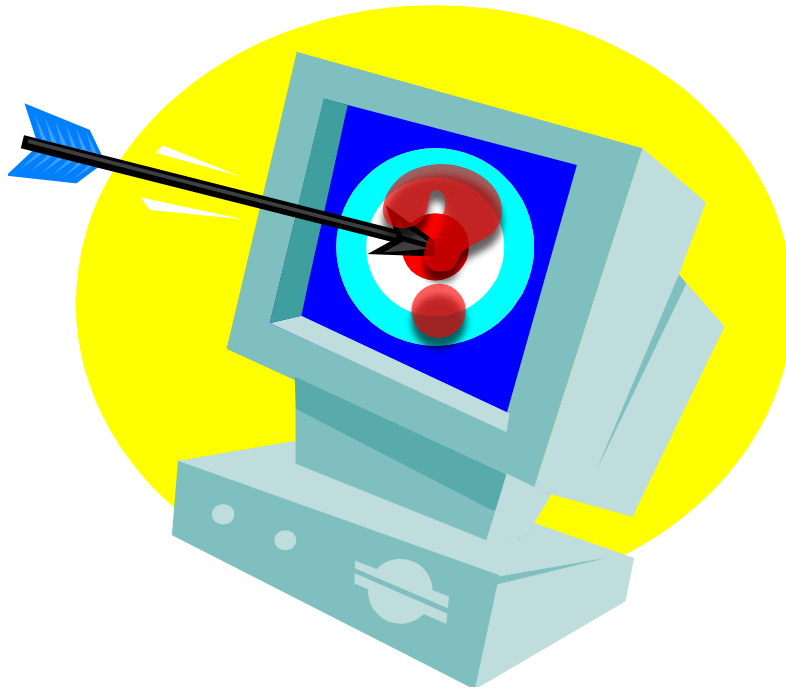
- Programmer's workflow
 - Execution Abstraction
 - Algorithmic theory – PRAM
 - Program (software) – XMTC
- **Compiler/Run-Time (middle-ware)**
- **Hardware/Simulator**

[CACM11]

Why is the toolchain important?

“A perfection of means, and confusion of aims, seems to be our main problem.”

Albert Einstein



XMTSim

The Cycle-Accurate Simulator of the XMT Architecture

- Overview and modeled components
- Software structure
- Discrete event vs. discrete time simulation
- Other features

Highlights



Image from wikipedia.

Highlights

- Highly configurable: Number of units, clock frequencies, ...

Highlights

- Highly configurable: Number of units, clock frequencies, ...
- Relatively easy to replace large components (i.e. ICN)

Highlights

- Highly configurable: Number of units, clock frequencies, ...
- Relatively easy to replace large components (i.e. ICN)
- Cycle-accuracy validated against the FPGA prototype

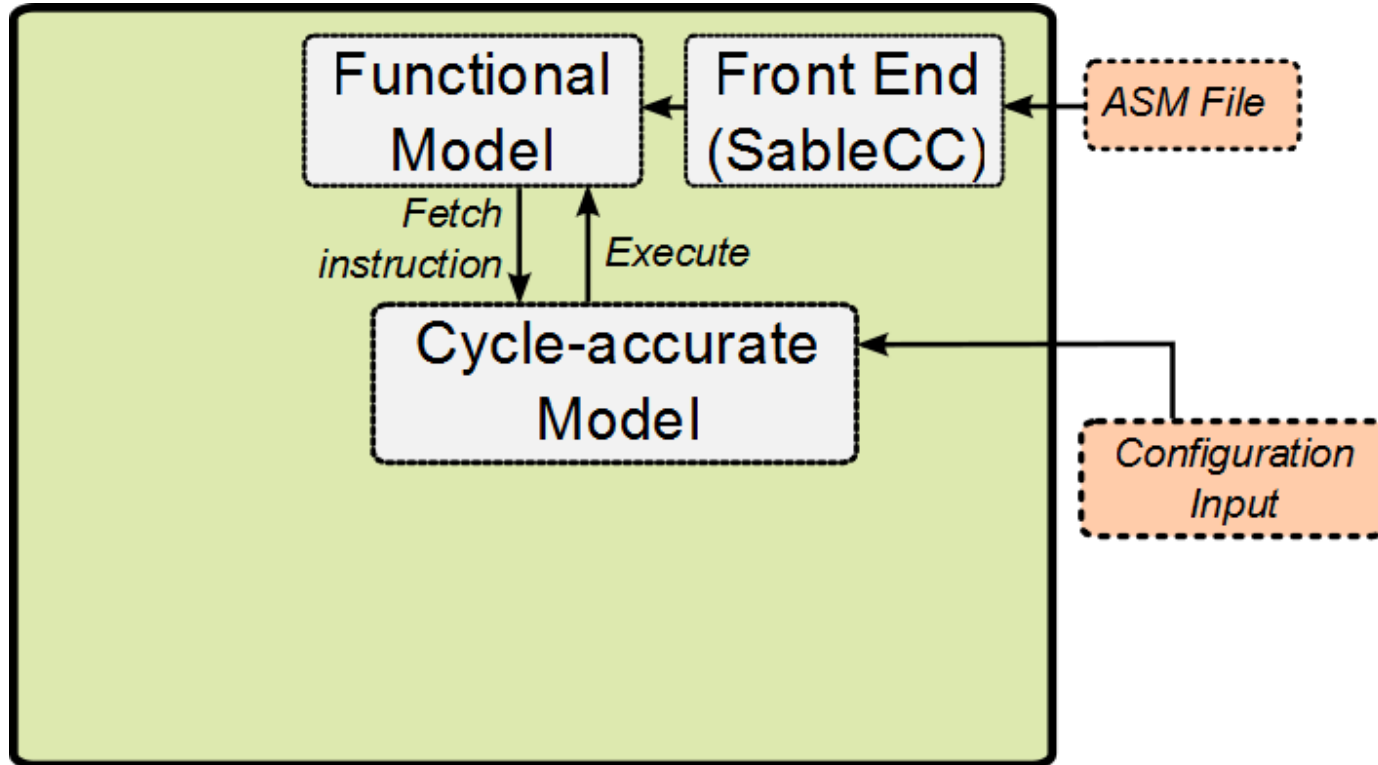
Highlights

- Highly configurable: Number of units, clock frequencies, ...
- Relatively easy to replace large components (i.e. ICN)
- Cycle-accuracy validated against the FPGA prototype
- Custom mechanisms for statistics collection, debugging, runtime modifications

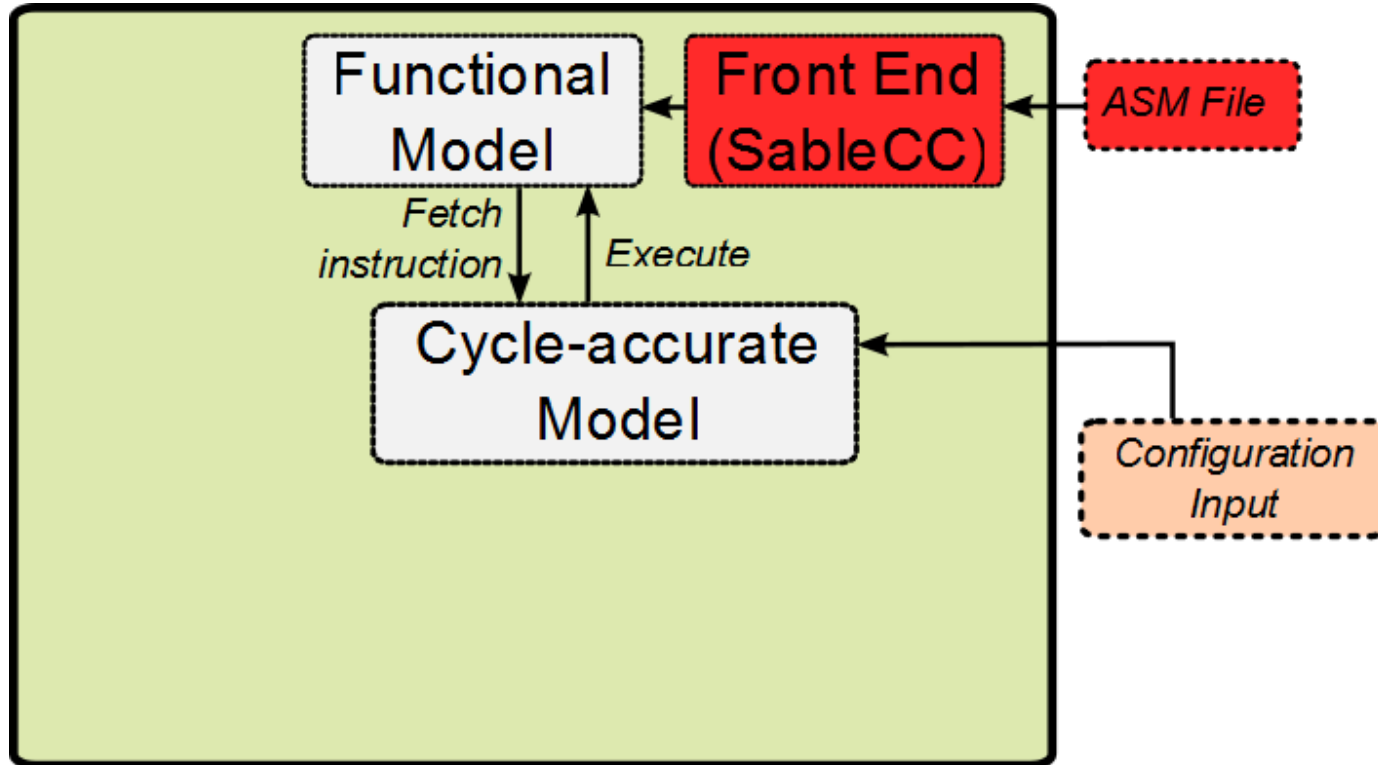
Implementation details

- Written in Java (object oriented)
 - ~28K lines of code + ~14K lines of inline documentation
- Simulation speed (Intel Xeon server @ 3GHz)
 - 10K – 2M instructions/second
 - Order of 1K cycles/second (1024-core conf.)

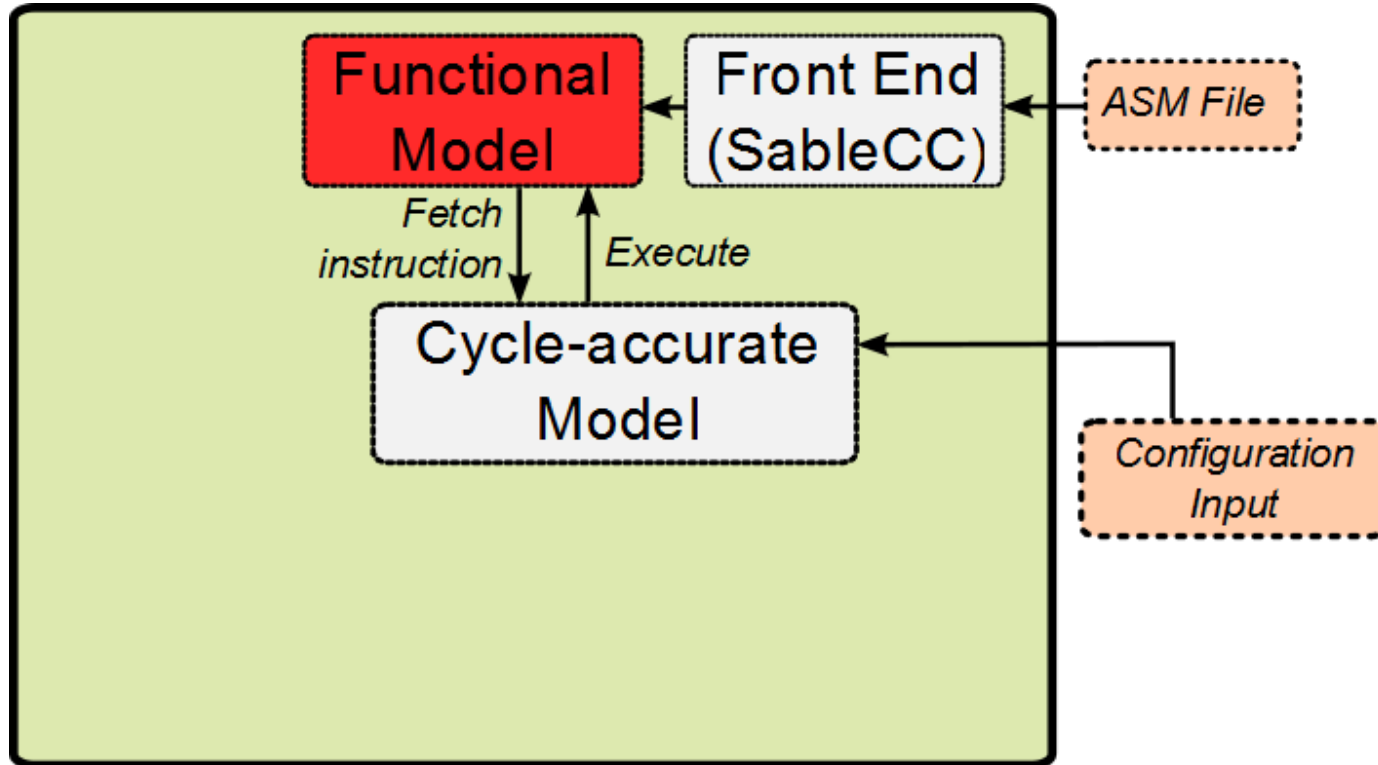
Software Structure



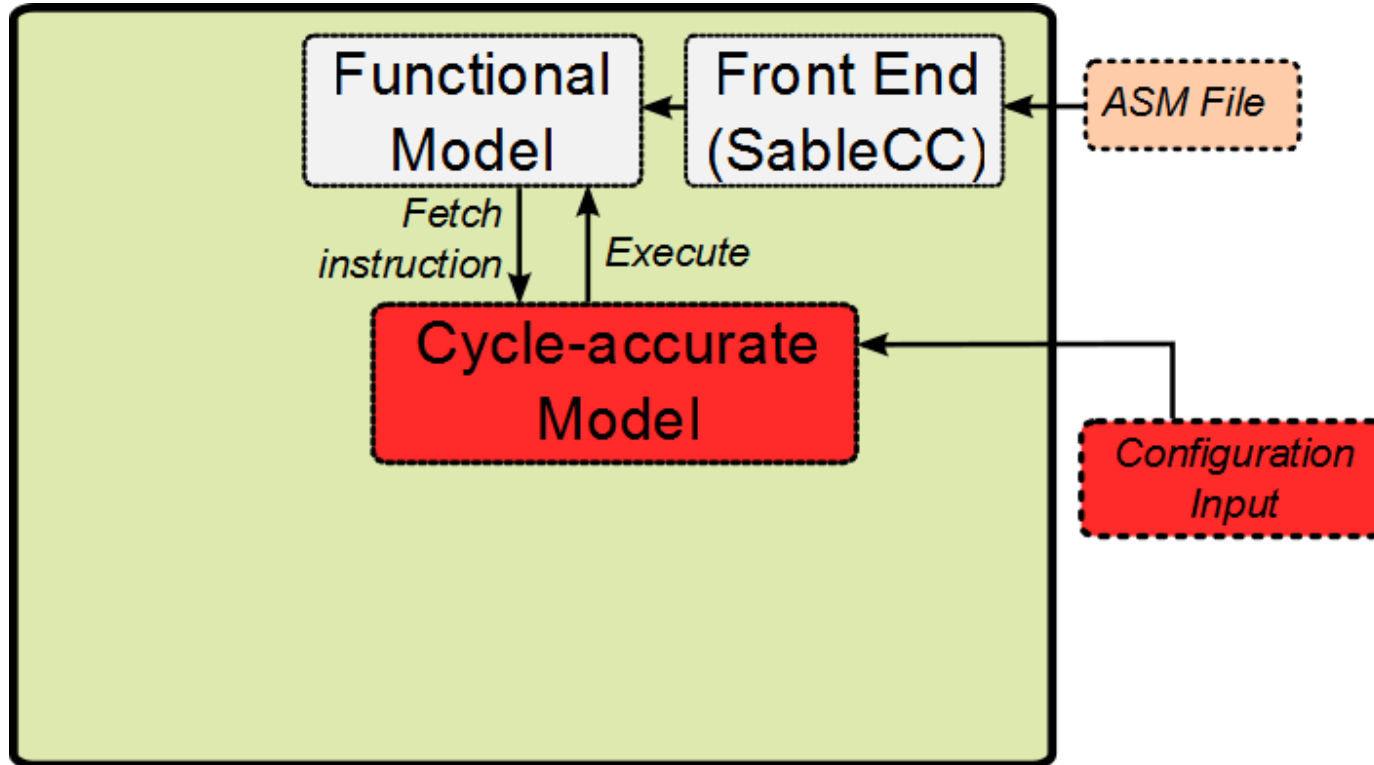
Software Structure



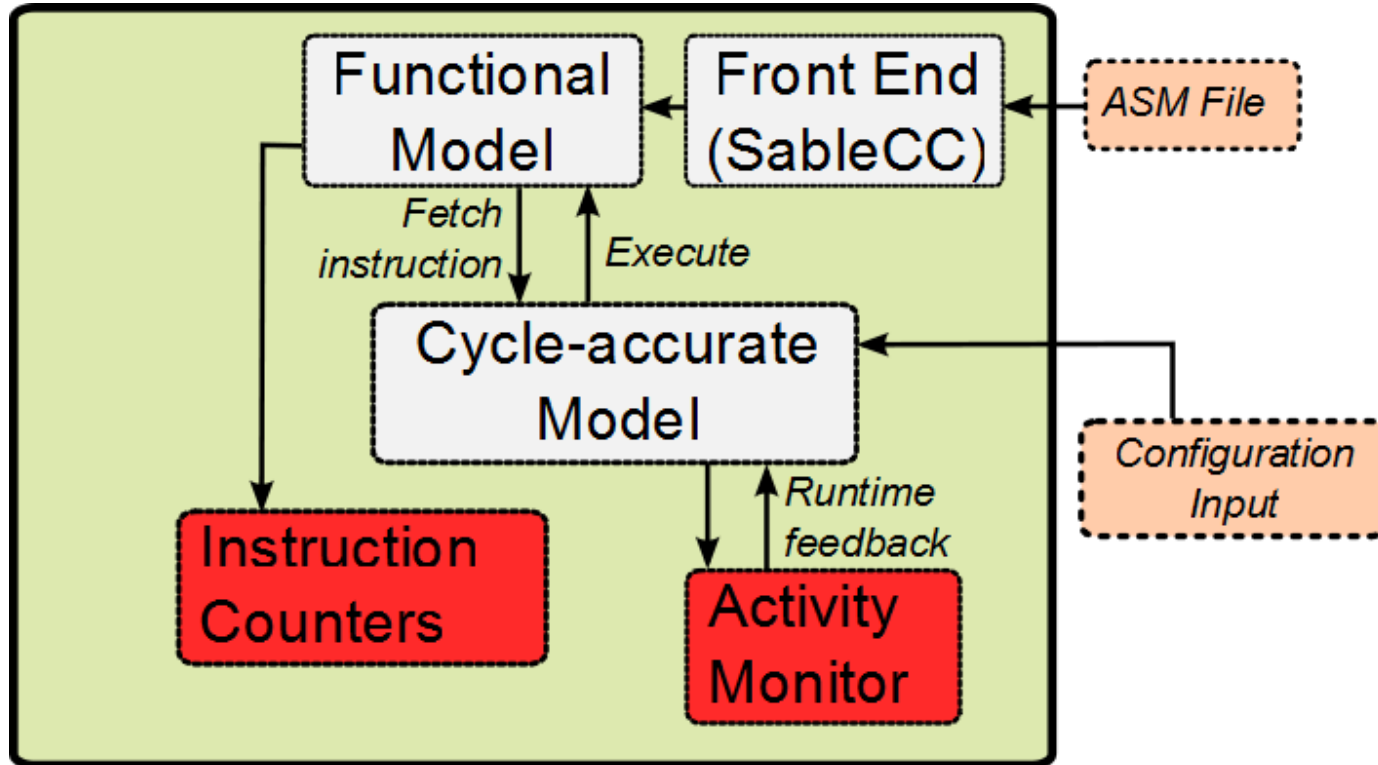
Software Structure



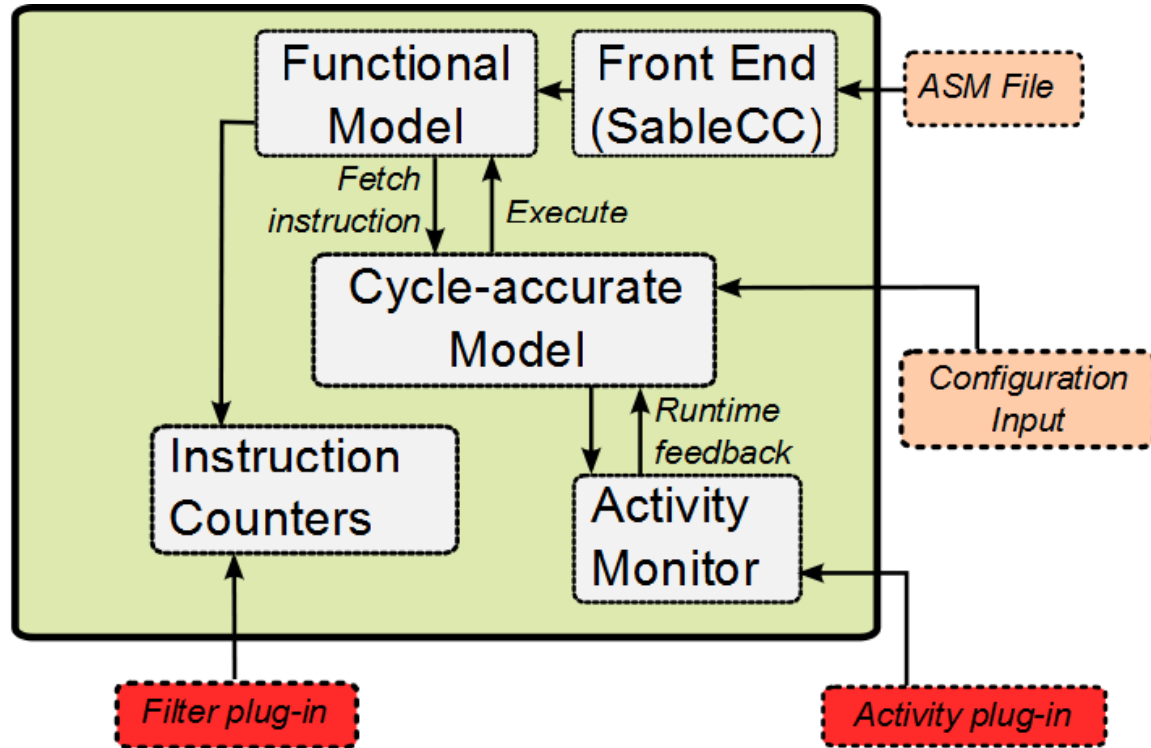
Software Structure



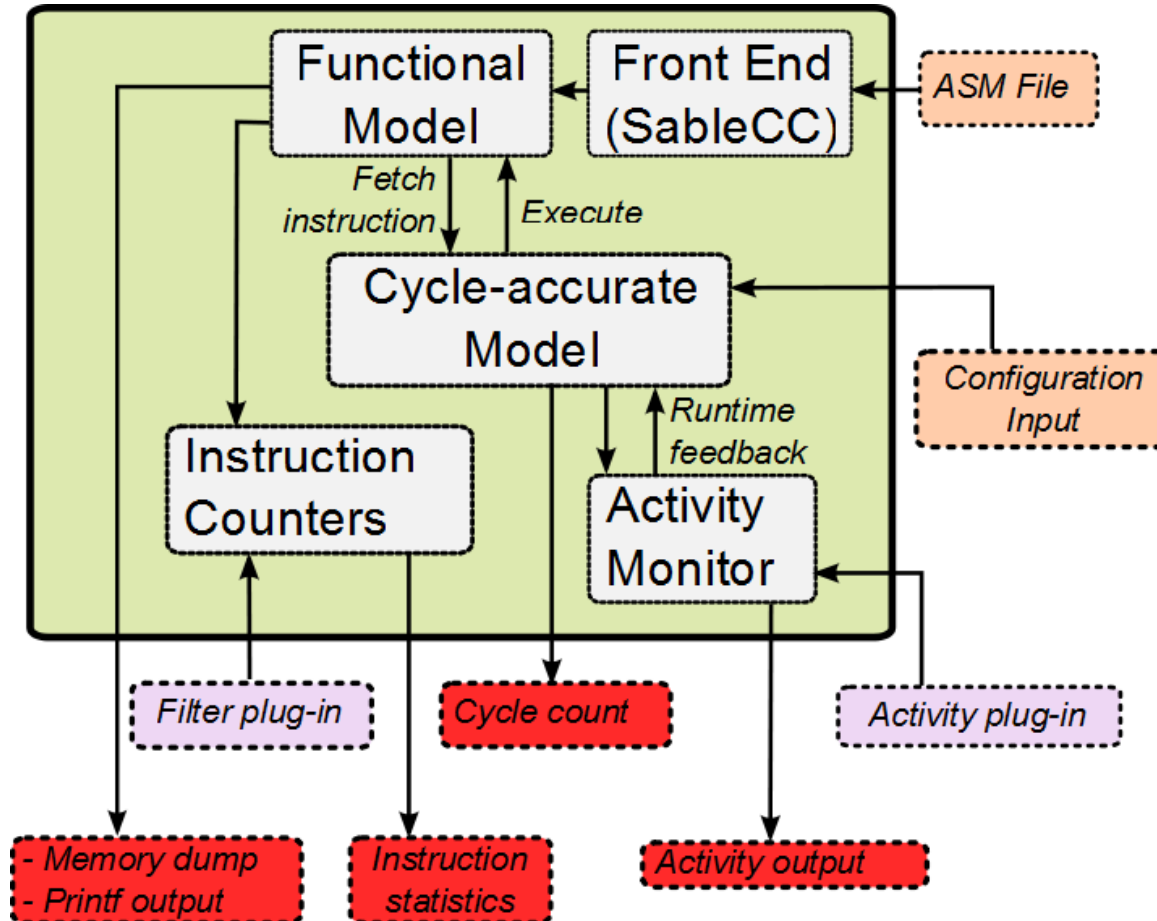
Software Structure



Software Structure



Software Structure



Simulation Strategy

Discrete Time vs. Discrete Event Simulation

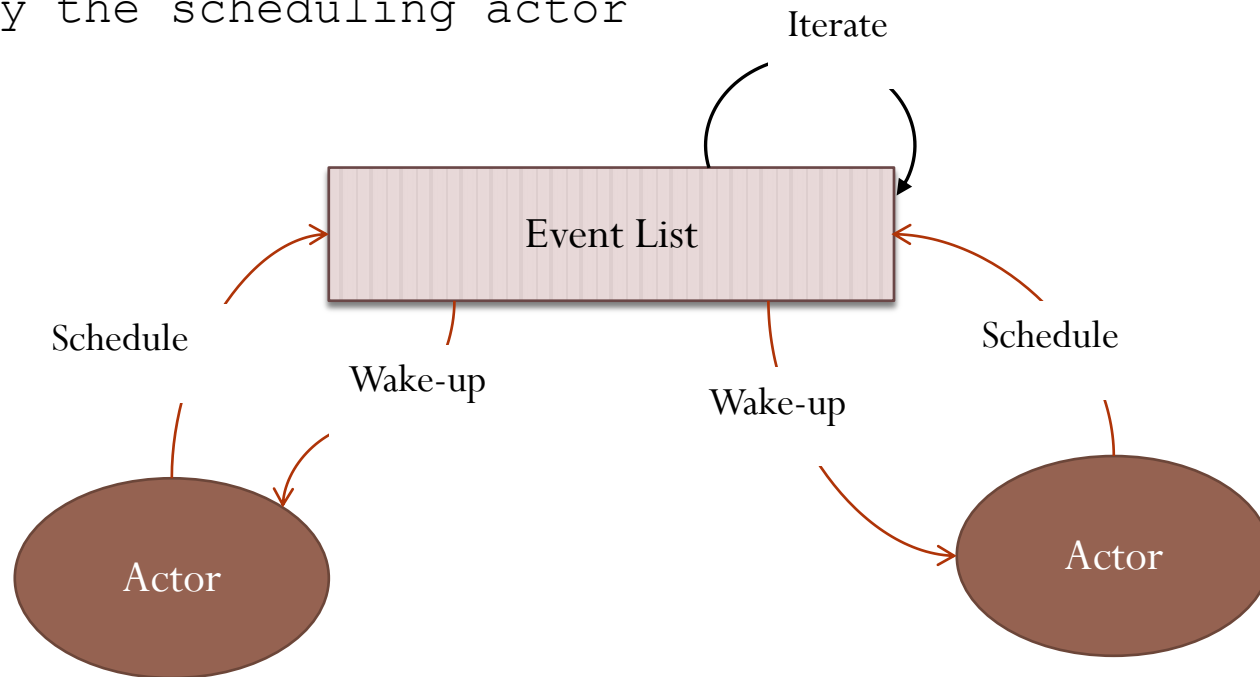
Discrete Time Simulation

Main loop:

```
while(still running) {  
    Set clock to next cycle  
    Execute simulation code for one cycle  
}
```

Discrete Event Simulation

```
while(still running) {  
  Set clock to next event time  
  Remove event from queue  
  Notify the scheduling actor  
}
```



DE vs. DT Simulation

Discrete Time Simulation

- More compact code for smaller simulations
- Efficient if a lot of work done for every simulated cycle

Discrete Event Simulation

- Naturally suitable for an object-oriented structure
- More flexible in quantization of simulated time
- Can simulate asynchronous logic

DE vs. DT Simulation

Discrete Time Simulation

- Efficient if a lot of work done for every simulated cycle
- More compact code for smaller simulations

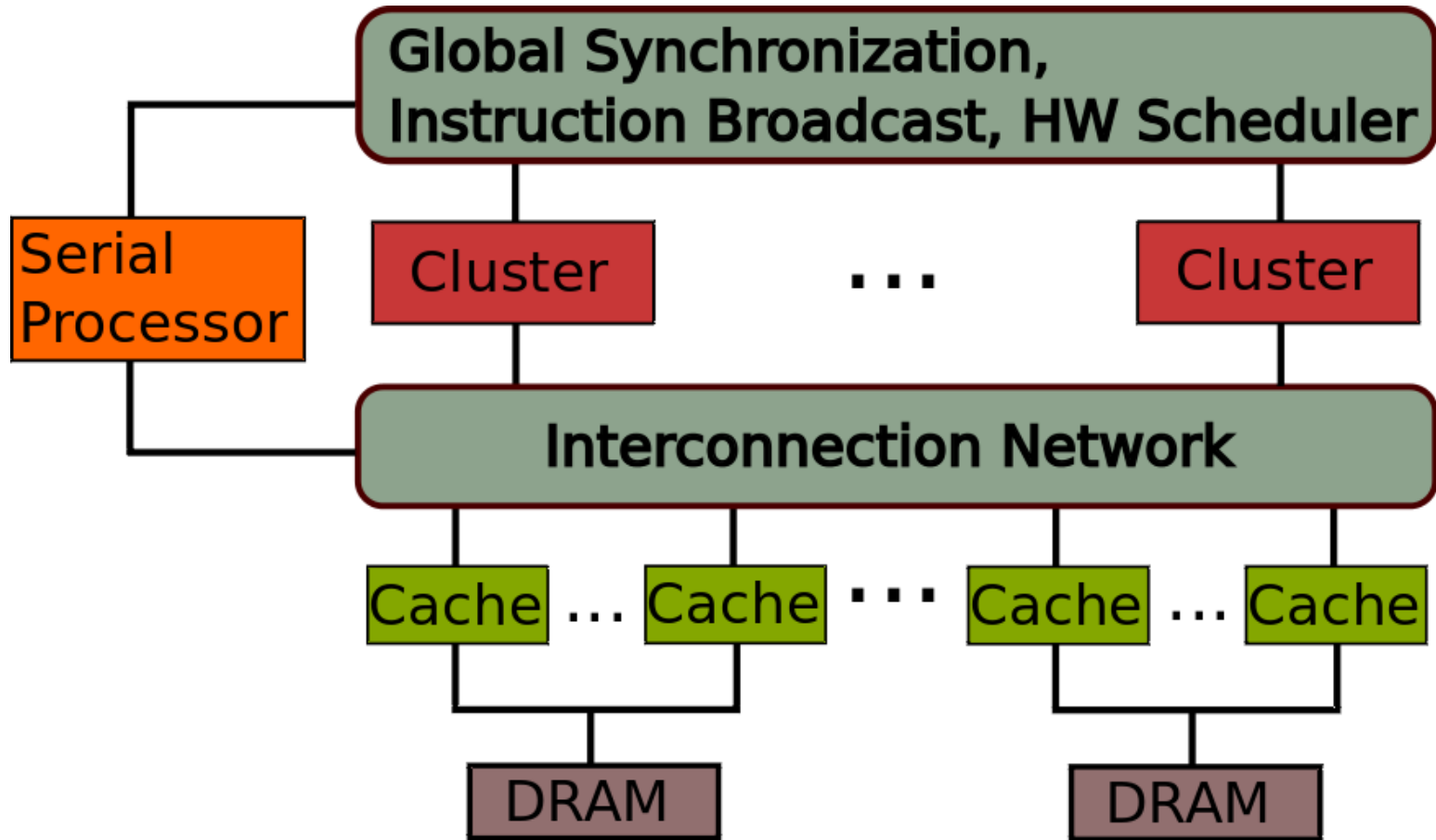
- Requires complex case analysis for a large simulator.
- Slow if not all components do work every clock cycle

Discrete Event Simulation

- Naturally suitable for an object-oriented structure
- More flexible in quantization of simulated time
- Can simulate asynchronous logic

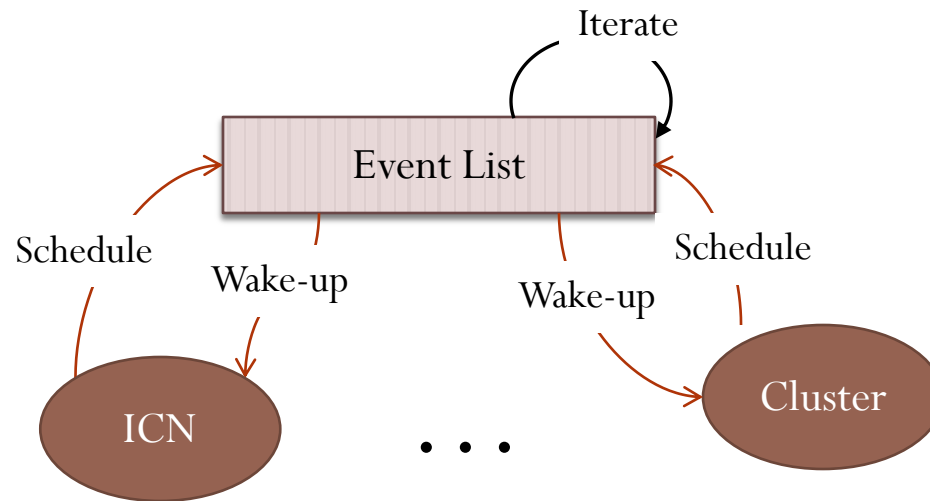
- Event list operations are expensive
- Might require more work for emulating one clock cycle

How do I simulate this?



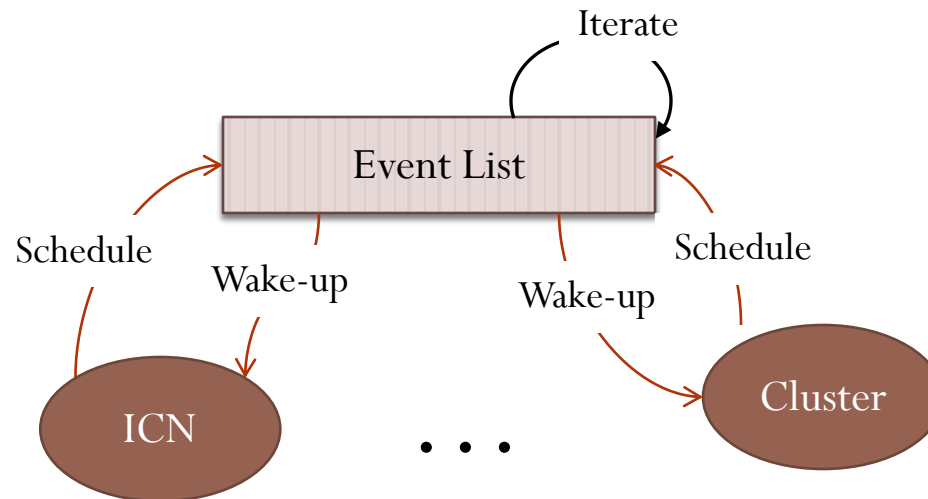
Best of both worlds

- Discrete event for a modular structure



Best of both worlds

- Heavy-weight action code for every actor (similar to the code for one iteration of DT simulation)



In the paper/tech. report

- Implementation details
 - Handling the communication between actors
 - Optimizing the event list
- Other features
 - Visualizing data on floorplan
 - Default plug-ins
 - Execution check-points

XMT Compiler

Optimizing compiler for explicit parallelism

- Serial GCC for parallel code
- Latency Hiding
- XMT Runtime
- Software structure of the compiler

Designing a compiler from scratch is hard...



How hire 1000s of developers for free?



Use GCC!

Serial GCC for parallel code?

- Illegal dataflow
- Reasons
 - Concurrent semantics
 - Control transfer between serial core and parallel cores

Example: Illegal Code Motion

XMTC Code

```
int A[N]=...;  
bool found=false;  
spawn (0, N-1) {  
    if(A[$]!=0)  
        found=true;  
}  
if (found) counter+=1;
```


Example: Illegal Code Motion

```
int A[N]=...;  
bool found=false;  
asm (spawn 0, N-1)  
    if(A[$]!=0)  
        found=true;  
asm (join)  
if (found) counter+=1;
```

Example: Illegal Code Motion

Compiler doesn't know the spawn construct!

```
int A[N]=...;  
bool found=false;  
asm (spawn 0, N-1  
    if(A[$]!=0  
        found=true;  
    asm (join)  
    if (found) counter+=1;
```



Example: Illegal Code Motion

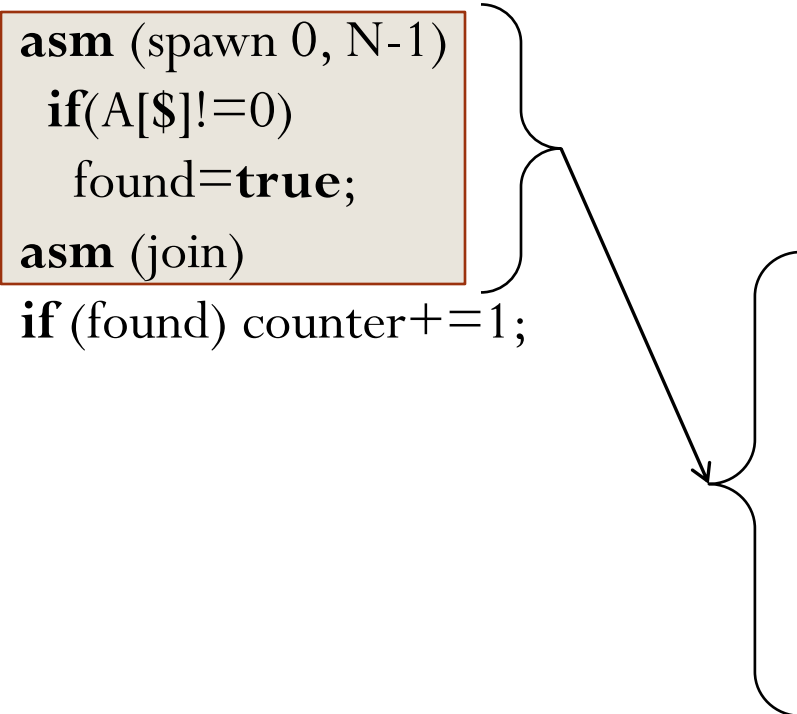
Compiler doesn't know the spawn construct!

```
int A[N]=...;  
bool found=false;  
asm (spawn 0, N-1)  
  if(A[$]!=0)  
    found=true;  
  if (found) counter+=1;  
asm (join)
```

A Partial Solution: Outlining

The compiler's view

```
int A[N]=...;  
bool found=false;  
asm (spawn 0, N-1)  
  if(A[$]!=0)  
    found=true;  
asm (join)  
if (found) counter+=1;
```



After Outlining

```
int A[N]=...;  
bool found=false;  
outlined(A,&found);  
if(found) counter+=1;  
-----
```

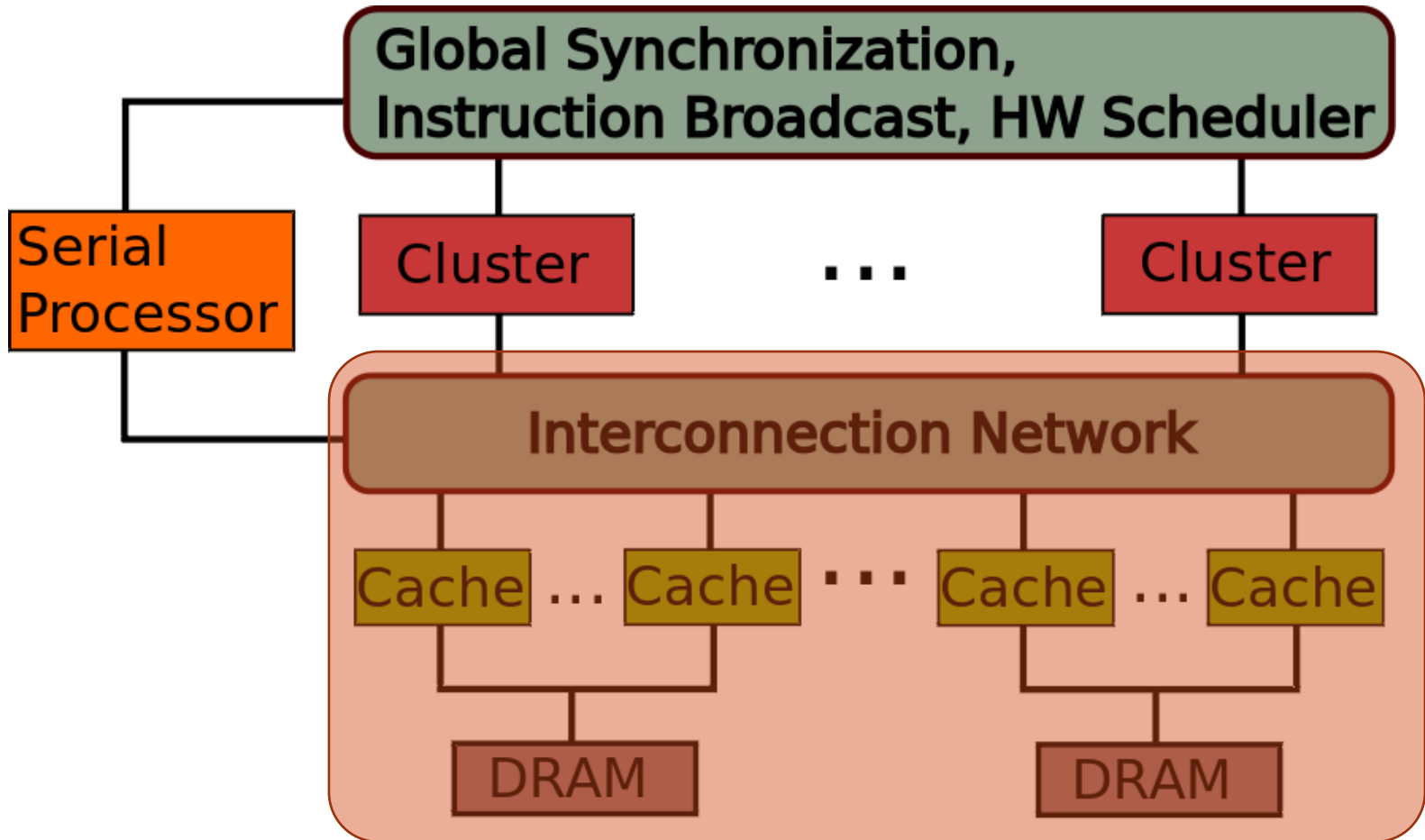
```
outlined(int (*A), bool *found) {  
  asm (spawn 0, N-1)  
  if (A[$]!=0)  
    (*found) = true;  
  asm (join)  
}
```

Other Issues

- Register Liveness Across MTCU → TCU transfers of control
 - Broadcast live MTCU registers
- Assembly code layout escaping spawn/join XMT statements
 - Analysis for relocating offending basic blocks

Details in the paper!

On-chip Memory Architecture



On-chip Memory Architecture

- Multiple shared cache memory modules
- No Coherent Private Cache
 - → Reduced power and ICN bandwidth traffic (+)
 - → Increased latency to d-Cache (-)
- But Hide Latency (*~30 cycles to shared caches*) with ...
 - Parallelism
 - Compiler optimizations

Latency Hiding

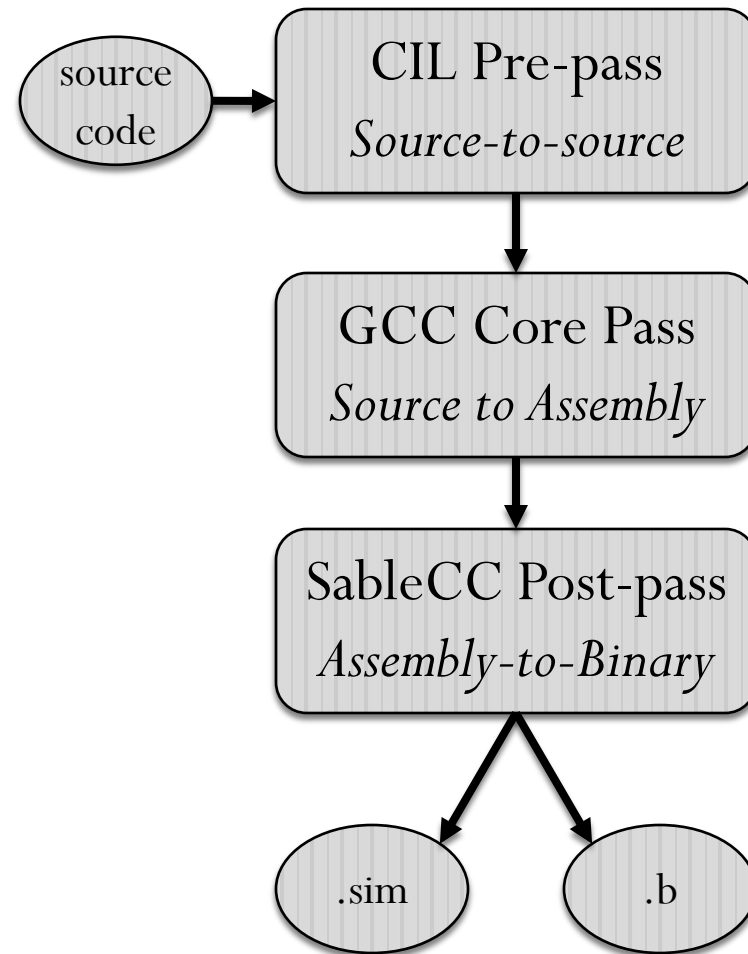
- Non-Blocking Stores
- Broadcasting of Code and Data
- Software Prefetching
- Planned for next release:
 - Read Only Caches (at Cluster Level)
 - Scratch Pads

Latency Hiding: Prefetching

- Loop Prefetching
- Unique resource aware prefetch algorithm
- Motivated by scarcity of TCU resources
 - 4 Prefetch buffer locations per TCU
- Existing algorithms do not take resources into account and assume that after prefetching all reads are cache hits

Software Structure of Compiler

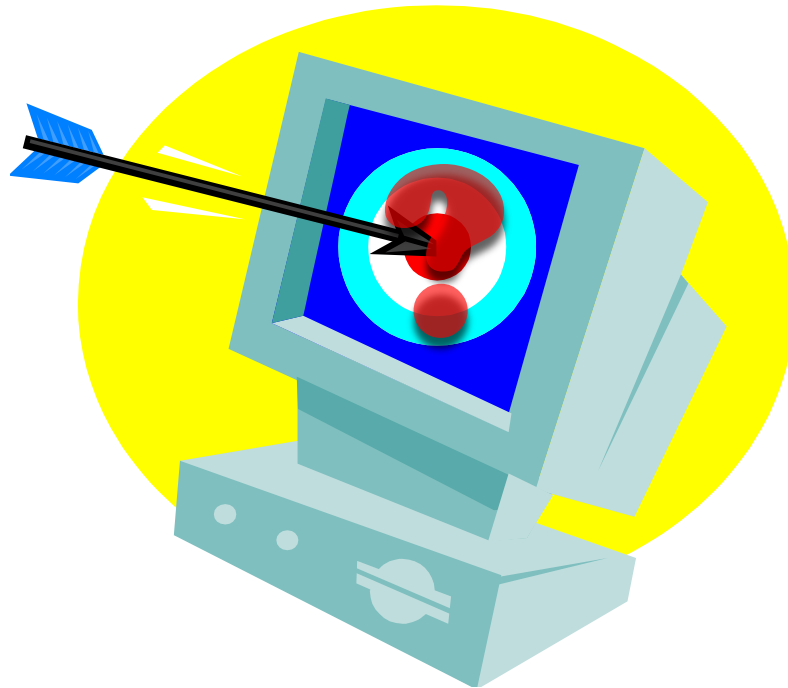
- CIL
 - Outlining
 - ...
- GCC
 - Broadcasting
 - Cactus-Stack Allocation
 - Prefetching
 - ...
- SableCC
 - Basic Block Layout
 - Linking
 - ...



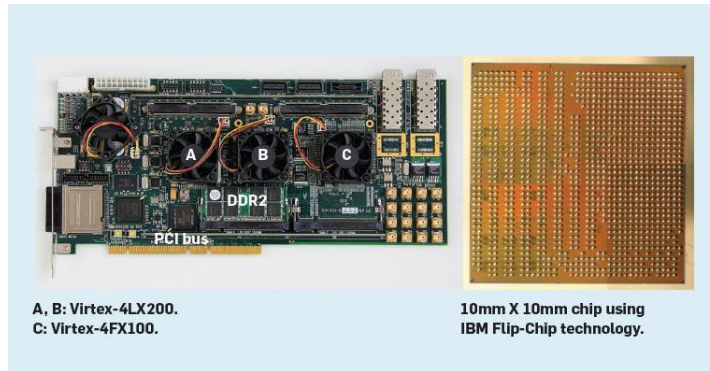
Why is the toolchain important?

“A perfection of means, and confusion of aims, seems to be our main problem.”

Albert Einstein



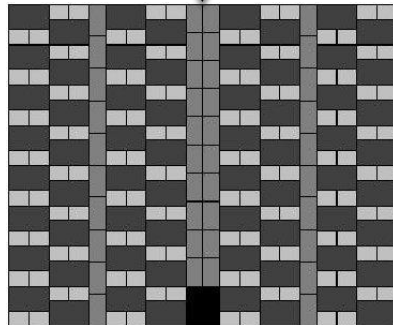
Current Implementation & Vision



64-core FPGA prototype
available

Figure from CACM11

Developing the specifications
via the toolchain!



1024-cores on-chip feasible
with today's technology
[HotPar2010]

How to advance towards the vision?

- Architecture/compiler research
 - XMT vs. GPU comparison [HotPar2010]
 - Scheduler for nested parallelism [PPoPP2010]
 - Software prefetching [IJPP2010]

How to advance towards the vision?

- Architecture/compiler research
 - Comparison with GPUs [HotPar2010]
 - Scheduler for nested parallelism [PPoPP2010]
 - Software prefetching [IJPP2010]
- Algorithms research
 - Max-Flow [SPAA11]
 - Bi-Connectivity [DIMACS2011]

How to advance towards the vision?

- Architecture/compiler research
 - Comparison with GPUs [HotPar2010]
 - Scheduler for nested parallelism [PPoPP2010]
 - Software prefetching [IJPP2010]
- Algorithms research
 - Max-Flow [SPAA11]
 - Bi-Connectivity [DIMACS2011]
- Teaching
 - Undergrad. and grad. classes – UMD/UIUC [EduPar2011]
 - High school/middle school [SIGCSE2010]

Under Release Testing

- Compiler
 - Parallel function calls
 - Scheduling for nested parallelism (lazy binary splitting)
- Simulator
 - Power/temperature models included (HotSpot from UVA)

Under Development

- Compiler
 - More latency hiding mechanisms
(read only caches and scratch pads)
- Simulator
 - Increasing simulator speed:
Parallel simulation, phase sampling and fast forwarding
 - Simulation of asynchronous interconnect in [TCAD2010]
- Operating system

Summary

- Explicit Multi-Threading
 - From abstraction/algorithms to hardware
- XMTSim
 - Highly configurable
 - Can simulate GALS, asynchronous logic, dynamic management
- XMT Compiler
 - How to use serial GCC for a parallel language?
 - Optimizations: Latency Hiding, efficient scheduling
- Impact
 - Architecture/compiler/algorithms research, teaching

Past Contributors

- Yoni Ashar, Thomas Dubois, Bryant Lee (UMD Students)
- Tali Moreshet, Katherine Bertaut (Swarthmore College)
- Genette Gill (Columbia University)

References

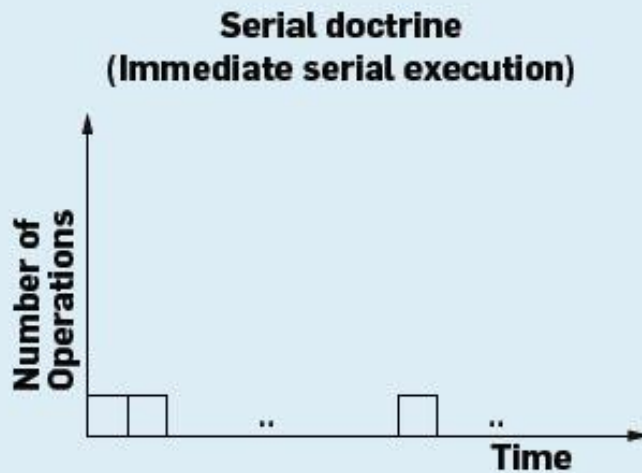
- [CACM2011] U. Vishkin. Using simple abstraction to reinvent computing for parallelism. CACM 54,1,p. 75-85, Jan. 2011
- [SPAA2011] G.C. Caragea and U. Vishkin. Better speedups for parallel max-flow. SPAA, 2011.
- [EduPar2011] D. Padua, U. Vishkin and J. Carver. Joint UIUC/UMD parallel algorithms/programming course. EduPar-11, in IPDPS, 2011
- [SIGCSE2010] S. Torbert, U. Vishkin, R. Tzur and D. Ellison. Is teaching parallel algorithmic thinking to high-school student possible? One teacher's experience. SIGCSE, 2010
- [IJPP2010] C.G. Caragea, A. Tzannes, F. Keceli, R. Barua and U. Vishkin. Resource-aware compiler prefetching for many-cores. Intern. J. of Parallel Programming, 2010
- [HotPar2010] C.G. Caragea, F. Keceli, A. Tzannes and U. Vishkin. General-purpose vs. GPU: Comparison of many-cores on irregular workloads. HotPar, 2010
- [PPoPP2010] A. Tzannes, G.C. Caragea, R. Barua and U. Vishkin. Lazy binary splitting: A run-time adaptive dynamic work-stealing scheduler. PPOPP, 2010.
- [DIMACS 2011] From asymptotic PRAM speedups to easy-to-obtain concrete XMT ones, invited talk, DIMACS Workshop on Parallelism: A 20-20 Vision, 2011
- [TCAD2010] M.N. Horak, S.M. Nowick, M. Carlberg and U. Vishkin. A low-overhead asynchronous interconnection network for GALS chip multiprocessor. TCAD p. 494-507, special Issue for NOCS, Apr. 2010

Backup Slides

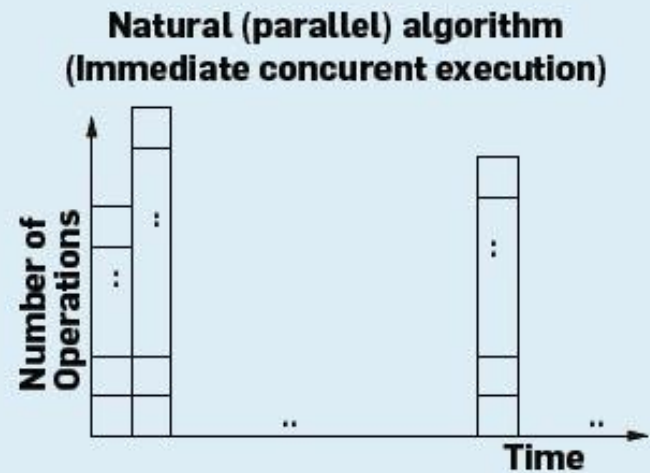
Speedups

- XMT vs. GTX280 GPU 6x speedup on irregular par.
- MaxFlow >100x over serial
- Bi-Connectivity Up to 33x over serial

Immediate Concurrent Execution



Time = Number of Operations



Time \ll Number of Operations

Figure from CACM11

How do actors communicate?

- Problem: Order of concurrent events is not deterministic
- Ports
 - Concept similar to HDLs/SystemC
- Event priority mechanism for ports
 - Two phases/priorities: Evaluate and update
- How to handle halt signals?
- Macro Actor contract: No combinatorial path between inputs and outputs

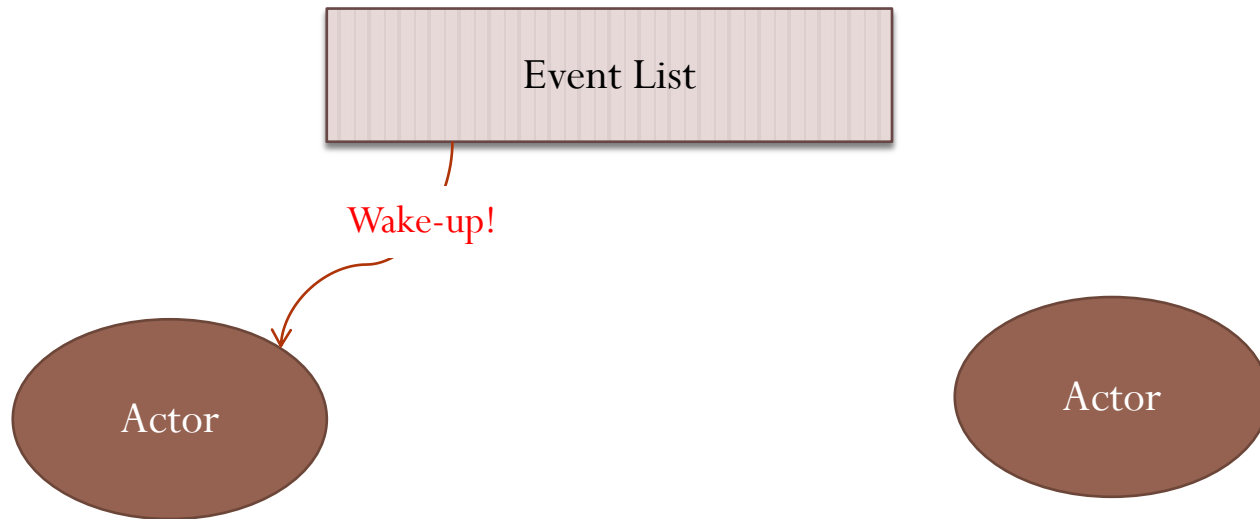
DE Simulation Example

Time = T



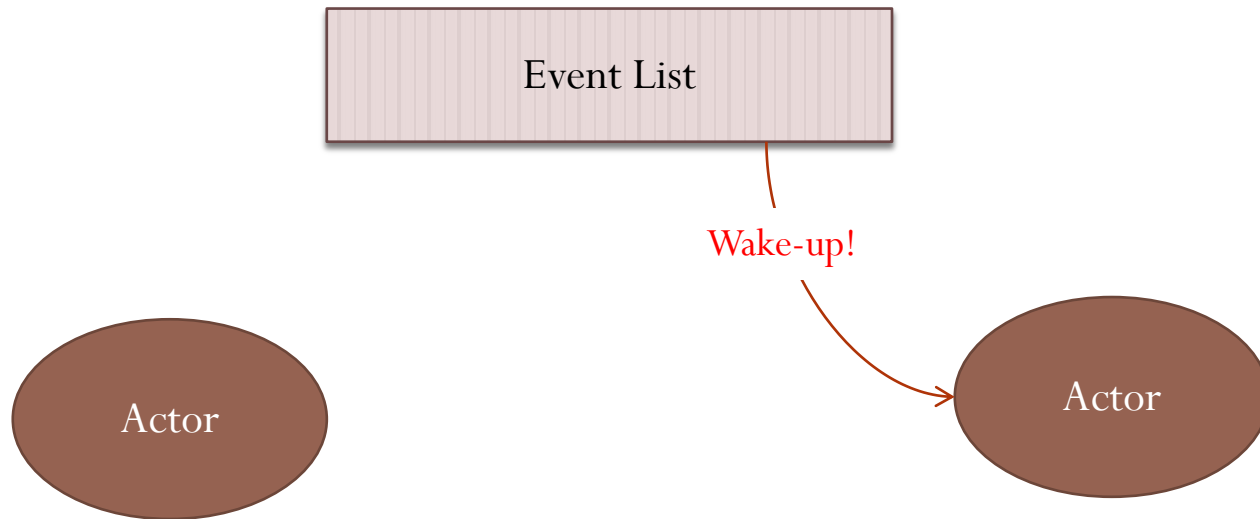
DE Simulation Example

Time = T+1

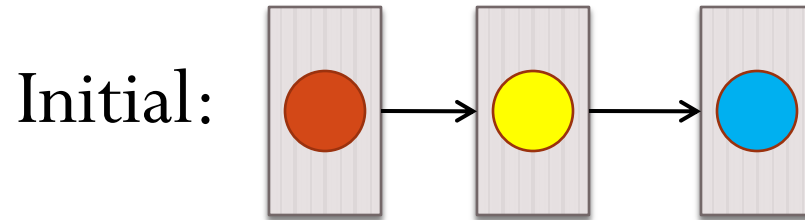


DE Simulation Example

Time = T+2



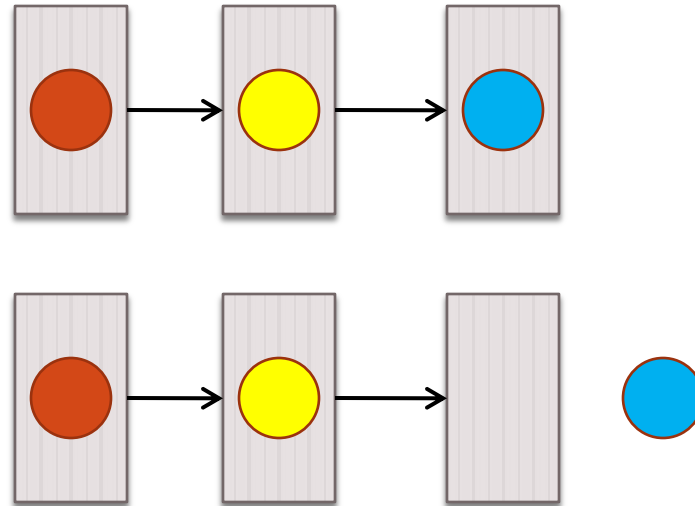
DT Example: Pipeline simulation



How to simulate advancing the pipeline for a single clock cycle?

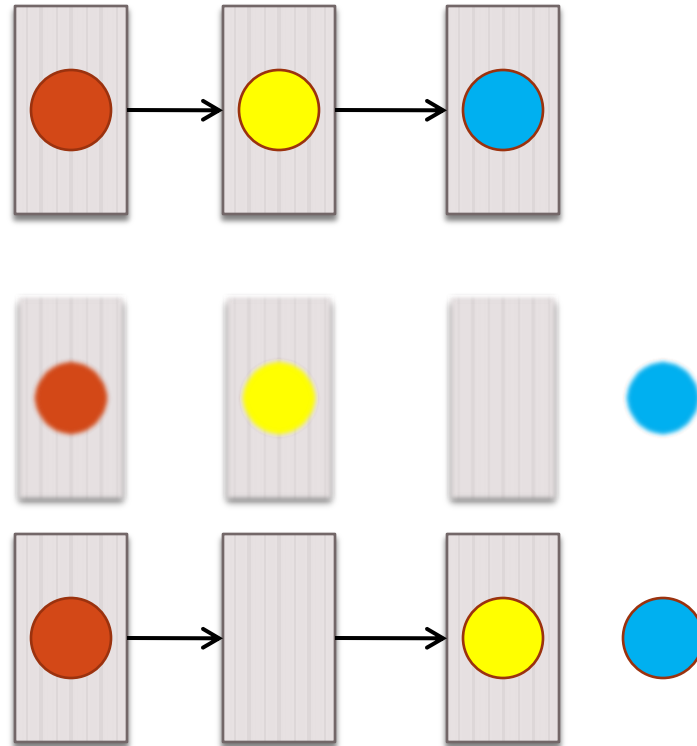
DT Example: Pipeline simulation

Initial:



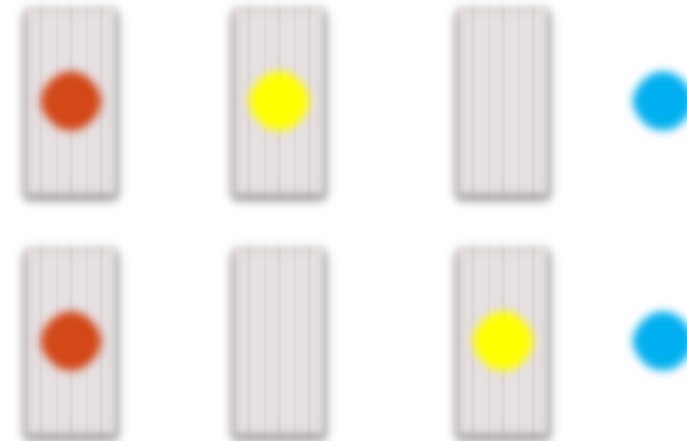
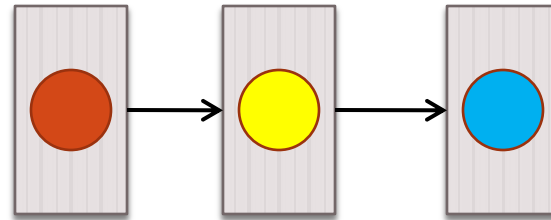
DT Example: Pipeline simulation

Initial:

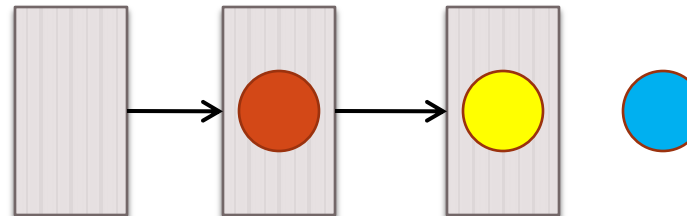


DT Example: Pipeline simulation

Initial:

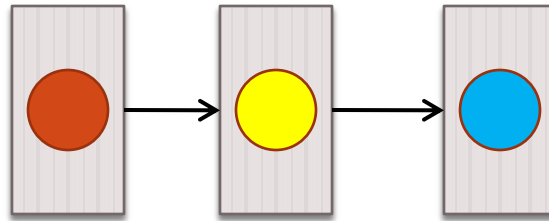


After 1 simulated cycle:



DE Example: Pipeline simulation

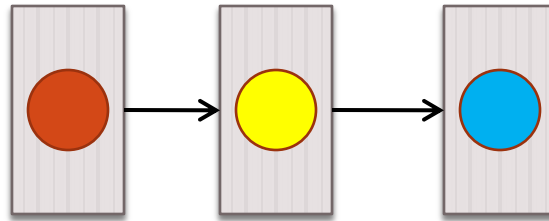
Initial:



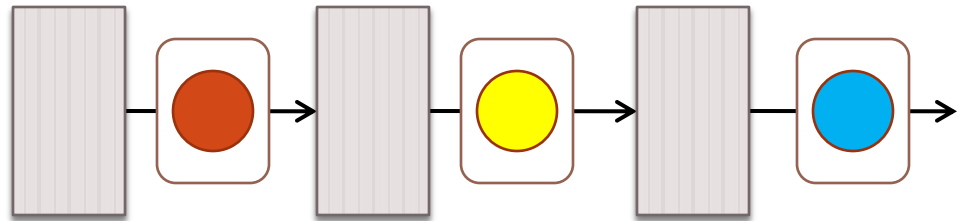
How to simulate advancing the pipeline for a single clock cycle?

DE Example: Pipeline simulation

Initial:

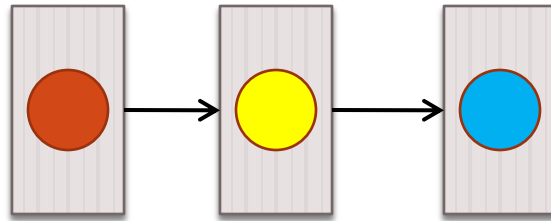


Use intermediate
storage:

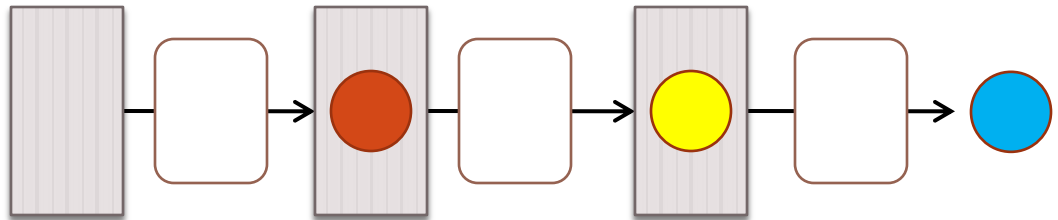


DE Example: Pipeline simulation

Initial:



After 1 simulated cycle:



XMTC Runtime

- Allocation of TCUs to virtual-threads
 - Using XMT hardware support (ps, chkid)
- Dynamic Memory Allocation
 - Serial only now. Parallel is left for future work
- Parallel Stack Allocation & Nested Parallelism Scheduling

XMTC memory model

Problem: Many processors read and write to the memory in parallel.

Question: In what order does processor A see memory operations from processor B? (Is the order experienced the same by A and B?)

Example: Initially: $x=0, y=0$;

Thread A

$x:=1$

$y:=1$

Thread B

Read y

Read x

Can B read $(x,y) = (1,0)$?

XMTC memory model

Initially: $x=0, y=0$

Thread A

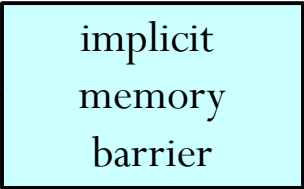
$x=1;$

$\text{psm}(1,y); //\text{atomic}\{y++\}$

Thread B

$\dots = \text{psm}(0,y); //\text{Read } y$

$\dots = x; // \text{Read } x$



implicit
memory
barrier