SCGPSim – A fast SystemC simulator on GPUs

Mahesh Nanjundappa, Hiren D. Patel, Bijoy A. Jose and Sandeep K. Shukla

This work was partially funded by an Air Force Labs, an AFOSR funding and NSF-CPA funding.

Overview of the presentation

- Motivation
- CUDA programming model
- Existing SystemC Simulator
- SystemC to CUDA
  - Extracting the Parallelism
  - Translator design flow
- Experimental Results
- Future plan

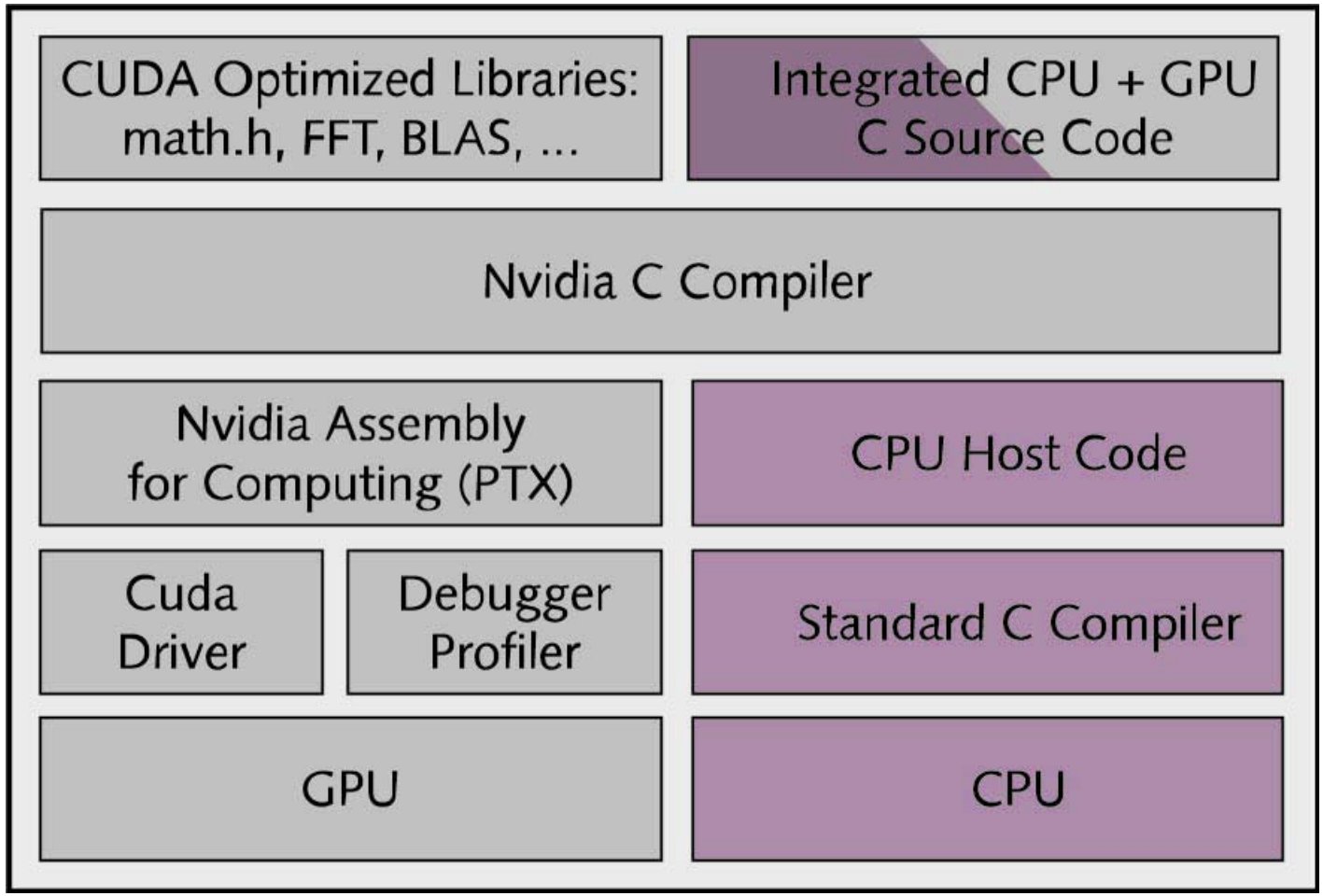
Motivation

- Functional simulation: main stay of verification
- A linear increase in complexity of designs causes an exponential increase to simulate it.
  - Ex: An addition of n bits, makes the total simulation cases increase by 2^n
- Discrete event simulators are suited for single core machines
  - DES on GPU was evaluated by Dr. Perumalla from Oak Ridge National Laboratory and he found that DES on GPU is slower than DES on CPU
  - But Cycle based simulator is faster in GPU when compared to CPU
- Cycle based simulators make full use of multi-core machines

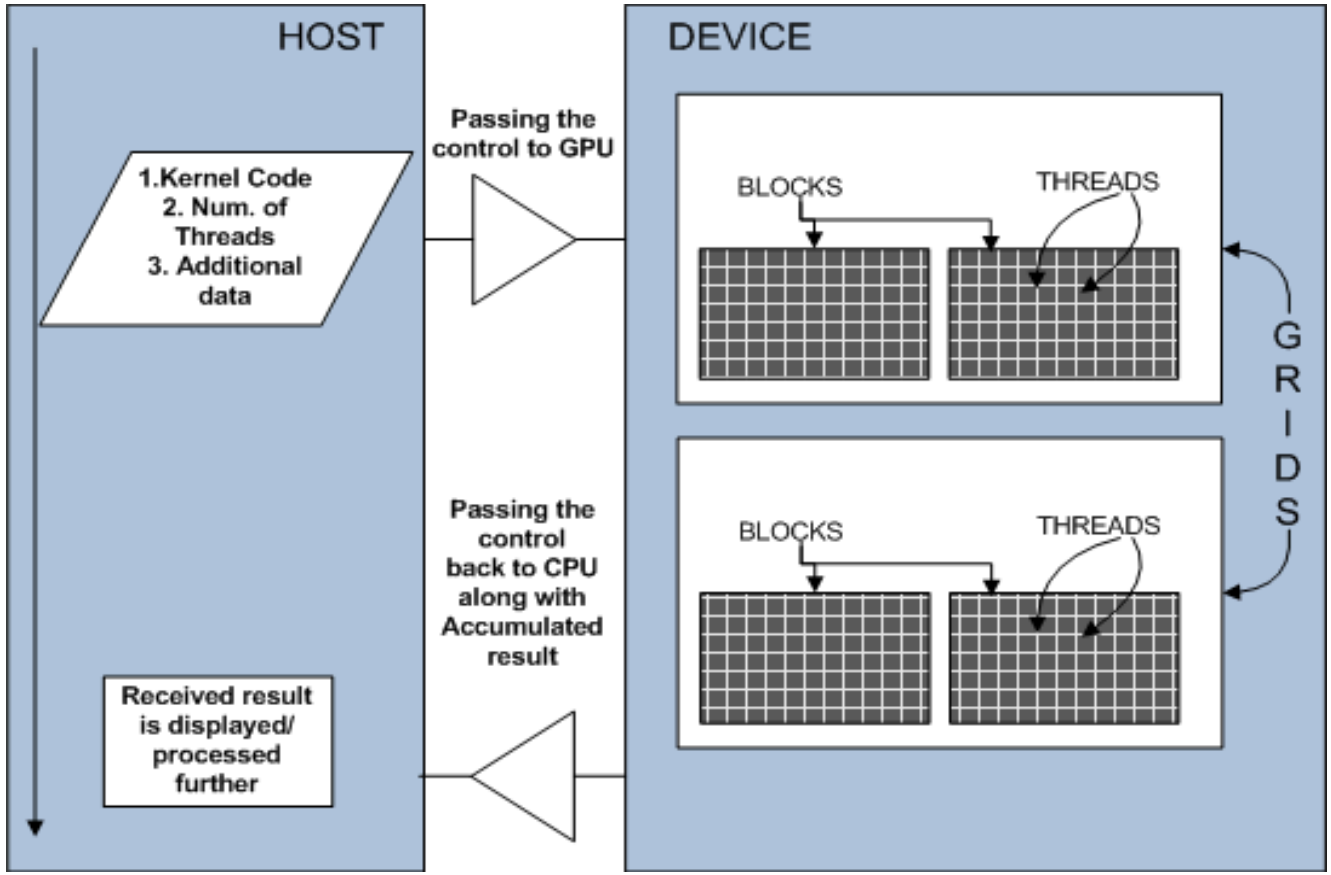
CUDA

- Extension to C to exploit processing power of GPU's
- Massively parallel architecture (can handle thousands of threads)
- Performance improvement is obtained by launching threads and making them execute part of application in parallel.
  - Ex: In a long code, parts of code (that can be executed in parallel) is made to execute by different threads

CUDA Platform

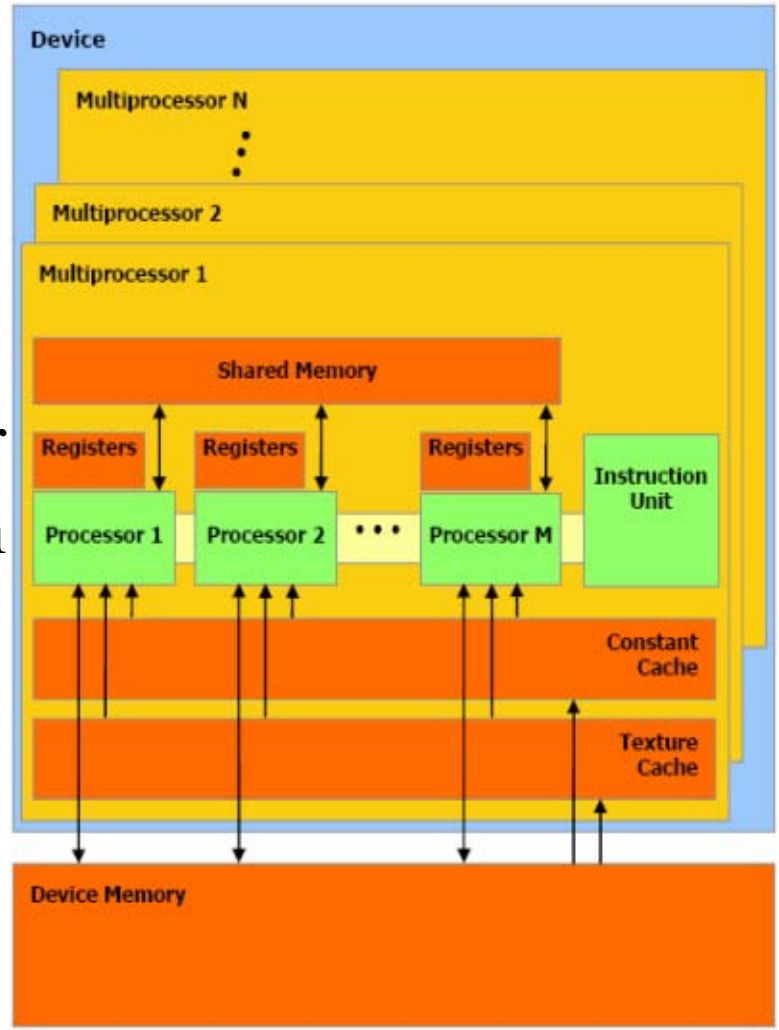


Programming Model



Hardware model

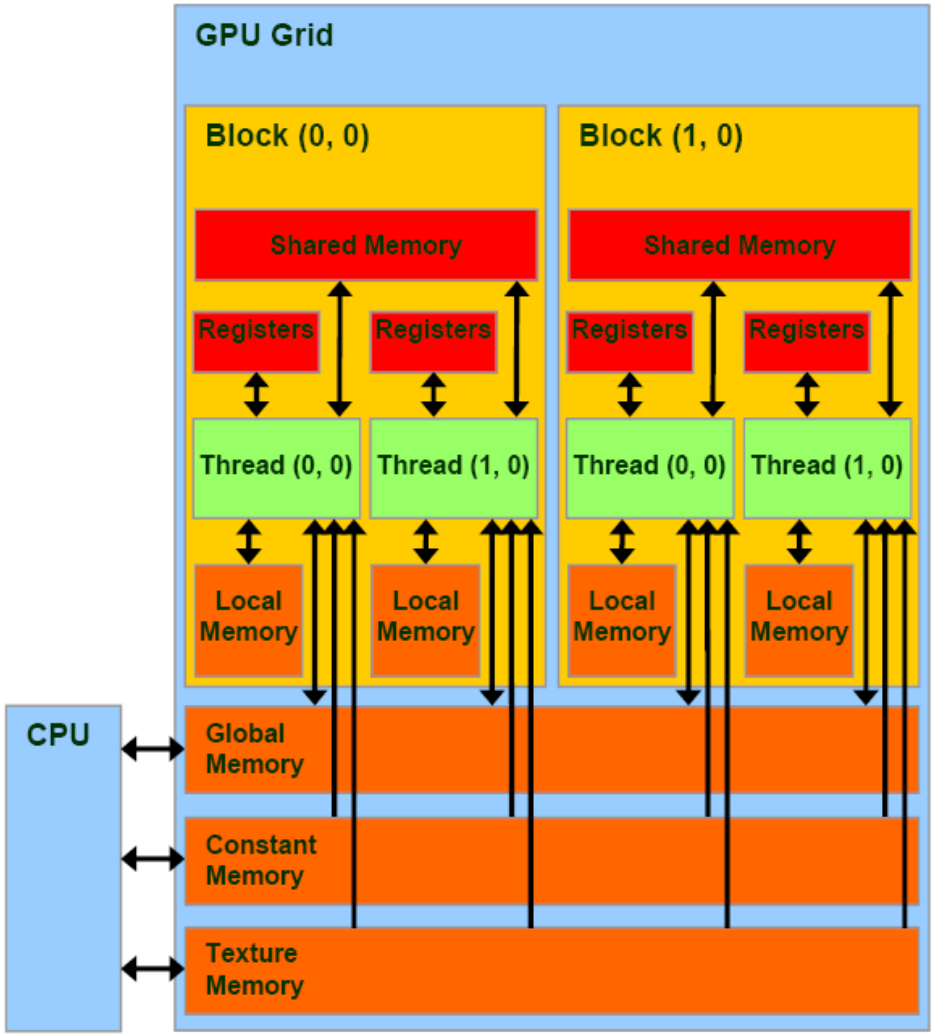
- A set of SIMD Multiprocessors
- At any given clock cycle each multiprocessor executes the same instruction but on different data
- Each multiprocessor has 8 smaller processors and each processor can handle 96 threads
- For 128 processors, 16 multi-processors and 12,288 threads



Memory model

- Read-write per-thread registers
- Read-write per-thread local memory
- Read-write per-block shared memory
- Read-write per-grid global memory
- Read-only per-grid constant memory
- Read-only per-grid texture memory

Memory model contd.



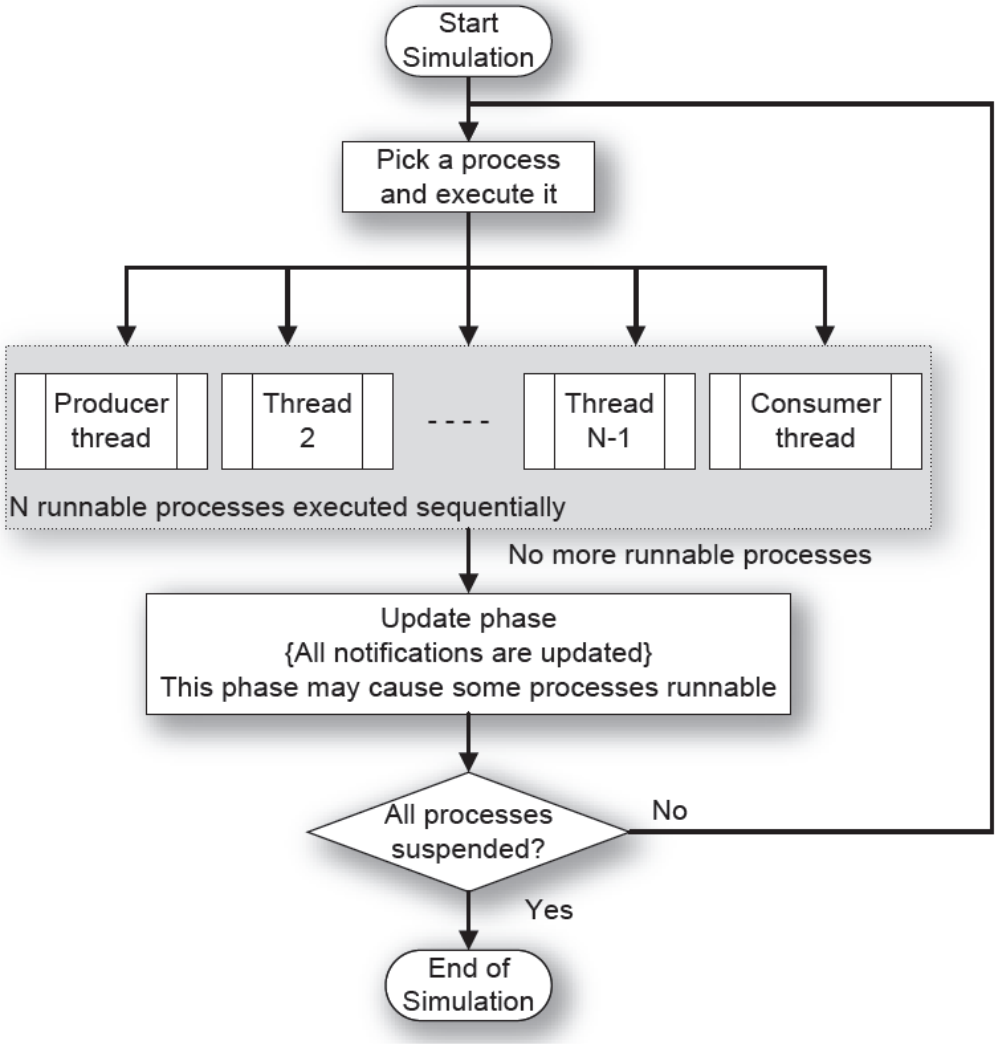
Thread Execution

- Threads are identified by *thread ID*
- 8 scalar processors per 1 multi-processor
- Threads that belong to one single block are executed by 1 multi-processor
- Synchronization between threads belonging to same block is achieved by single instruction __syncthreads()
- No means of synchronization of threads belonging to different blocks

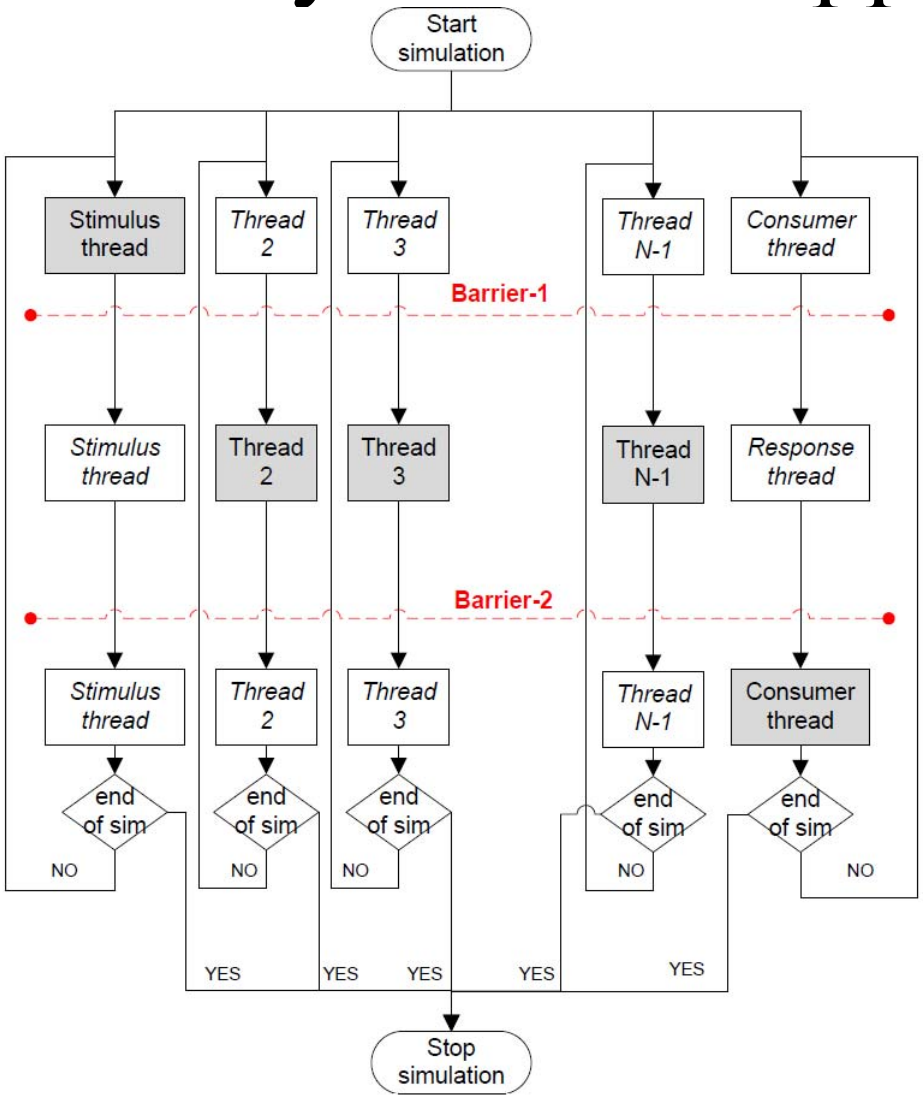
Existing SystemC simulator

- A discrete event simulator (DES)
- Simulator is built using application level thread libraries (co-operative threads)
- Operating system cannot schedule co-operative threads on multiple processors
- Hence it creates only one thread for any design
- Not utilizing the advantages of multi-core platforms

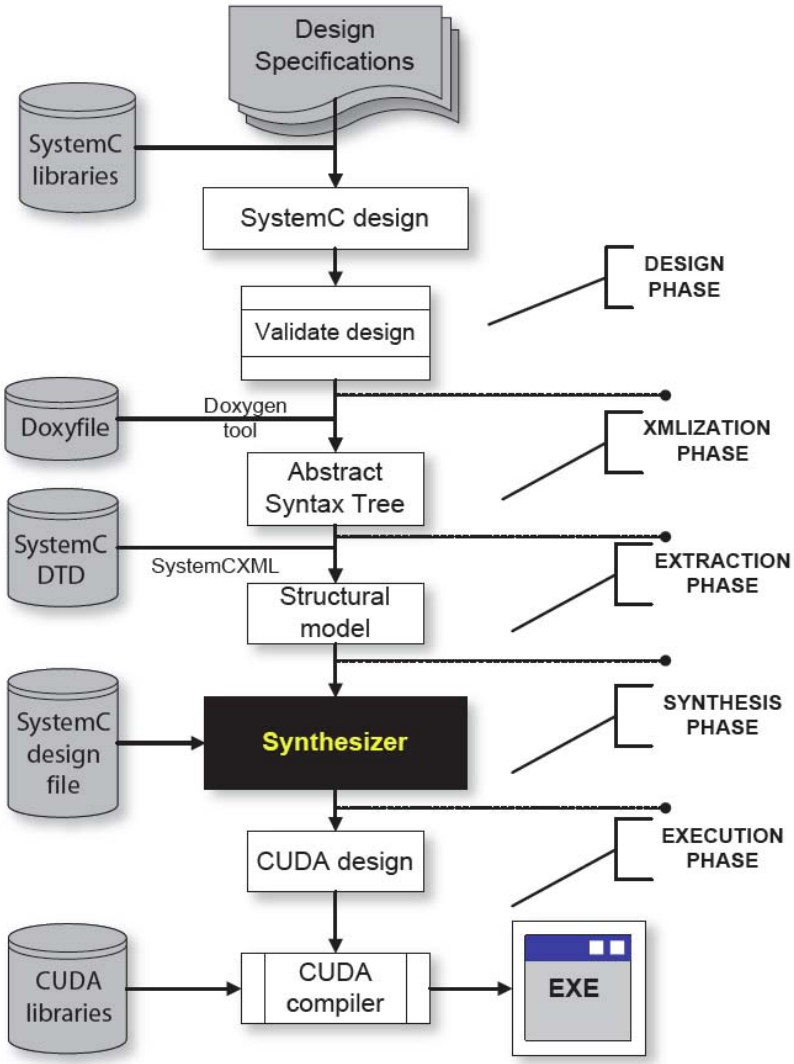
Execution in DES



Proposed Cycle based approach



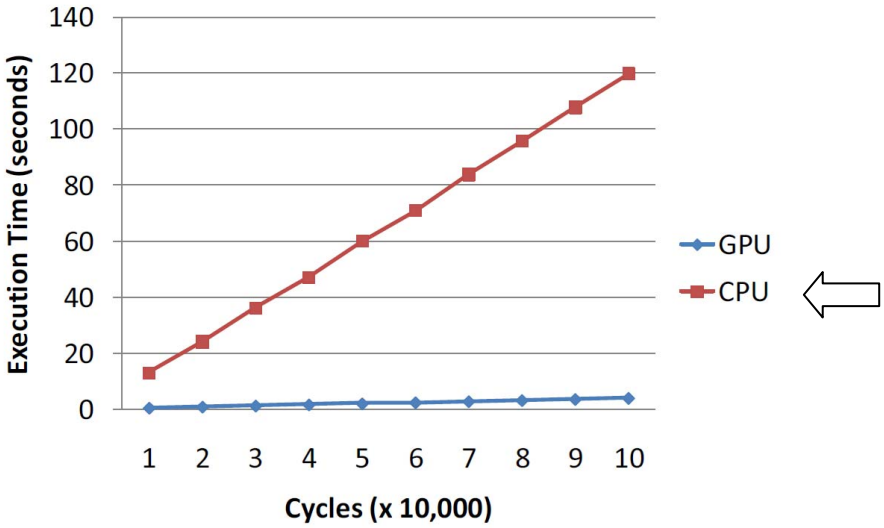
Design flow of SCGPSim



Experimental results

Table 1: Experimental Results

Design	CPU (sec)	CPU + GPU(sec)	SPEED UP
PIPELINE AES	120.1	3.916	30.66
FIR	51.9	1.37	37.88
10 STAGE BUFFER	28	0.277	101.083
3 STAGE BUFFER	11	0.276	39.85
SIMPLE ALU	13	0.146	89.041



AES execution time comparison for varying number of cycles

Future plan

- Provide full fledged formal proof
- A stable and effective Intermediate Representation (IR) for SystemC
- Automatize the translation of synthesizable SystemC to Intel-TBB so that we can make use of Multicore machines and Larrabee platform
- Extending the concept for System Verilog, RTL and others

Questions??

Thanks for your time.

BACK UP

Good about CUDA

- Huge number of threads can be executed in parallel
- Once work is transferred to GPU, CPU is free to do its work.
- Fast shared memory apart from cache which plays an important role
- Integer, bitwise and floating point operations are supported.
- Atomic operations are supported in GPU's with compute capability >1.0

Bad about CUDA

- Recursion free (no recursive functions) subset of C
- No recursive kernel calls (GPU cannot create threads by itself)
- No Object oriented programming and No I/O's
- Synching is only allowed between threads of same block
- The bus bandwidth and latency between the CPU and the GPU may be a bottleneck
- Even though on 12000 odd threads can be created, effective parallelism will be achieved only if threads are made to work in different processors
- Limitation for any inherently divergent task
- Global memory read/write is costly

Algorithm 1: SCGPSim synthesizer: Translating SystemC Design to CUDA Design

```
   /* Collect all signals.                                          */
 1 S_all ← ⋃_{m∈M} m.S
   /* Convert all signals.                                          */
 2 writeVariableBuffer(c.S^c, S_all)
   /* Generate CUDA kernel.                                         */
 3 for m ∈ M do
       /* Enter loop only if m ∈ F_t                                */
 4     for f ∈ m.F_t do
 5         b_n ← head(f), n ← 0, loc ← 0
           /* Generate thread code.                                 */
 6         while isWhileTrue(b_n) ≠ true do
 7             c.f^c ← c.f^c • checkCompatible(b_n)
 8             n ← n + 1
 9         end
10         c.f^c ← c.f^c • writeSwitchBeginBlock()
11         c.f^c ←
           c.f^c • writeFirstCaseBeginBlock(loc)
12         while n ≤ |f| − 1 do
13             if isWait(b_n) then
14                 c.f^c ← c.f^c • writeCaseEndBlock(loc)
15                 loc ← loc + 1
16                 c.f^c ←
                   c.f^c • writeNewCaseBeginBlock(loc)
17             else
18                 c.f^c ← c.f^c • checkCompatible(b_n)
19             end
20             n ← n + 1
21         end
22         c.f^c ← c.f^c • writeSwitchEndBlock()
23     end
24     writeMethods(m.F_m, c)
25 end
   /* Output the CUDA program                                       */
26 output(c)
```