

Simulating GPGPUs

ESESC Tutorial

Speaker: Alamelu Sankaranarayanan



*Department of Computer Engineering,
University of California, Santa Cruz*
<http://masc.soe.ucsc.edu>



Outline

- Background
- GPU Emulation Setup
- GPU Simulation Setup
- Running a GPGPU application

The Landscape Today

- *Heterogeneous Computing* : an alternate Paradigm
- GPUs are being increasingly used to augment CPU cores
 - Popularity of programming languages like CUDA / OpenCL
 - Application in Computer Vision & Image Processing , Augmented reality, Big Data, Machine Learning, etc.

The Landscape Today

- More computational capability with each new GPU
 - Increasing processing elements with each new generation
- Tighter coupling of the CPU and GPU
 - AMD's APUs, HSA
- Mobile / Embedded applications
 - Emphasis on energy efficiency
- Newer processor architectures like Knights Corner

Expectations from a simulator

- More computational capability with each new GPU
 - Increasing processing elements with each new generation
 - Tighter coupling of the CPU and GPU
 - AMD's APUs, HSA
 - Mobile / Embedded
 - Emphasis on energy efficiency
 - Newer processor architectures
- **More PEs → More threads → Longer Simulation Times**
 - **FAST simulators needed!**
 - **Ability to easily vary the architectural specifications like number of PEs, memory subsystem configuration, Allowable threads, Divergence mechanisms etc.**

Expectations from a simulator

- More computational capability with each new GPU
 - Increasing processing elements with each new generation
- Tighter coupling of the CPU and GPU
 - AMD's APUs, HSA
 - Ability to model a heterogeneous system with both CPUs and GPUs
- Mobile / Embedded applications
 - Emphasis on energy efficiency
- Newer processor architectures like Knights Corner

Expectations from a simulator

- More computational capability with each new GPU
 - Increasing processing elements with each new generation
- Tighter coupling of the CPU and GPU
 - AMD's APUs, HSA
- Mobile / Embedded applications
 - Emphasis on energy efficiency
- Newer processor architectures like Knights Corner

- Integrated Power Model
- Thermal?

Expectations from a simulator

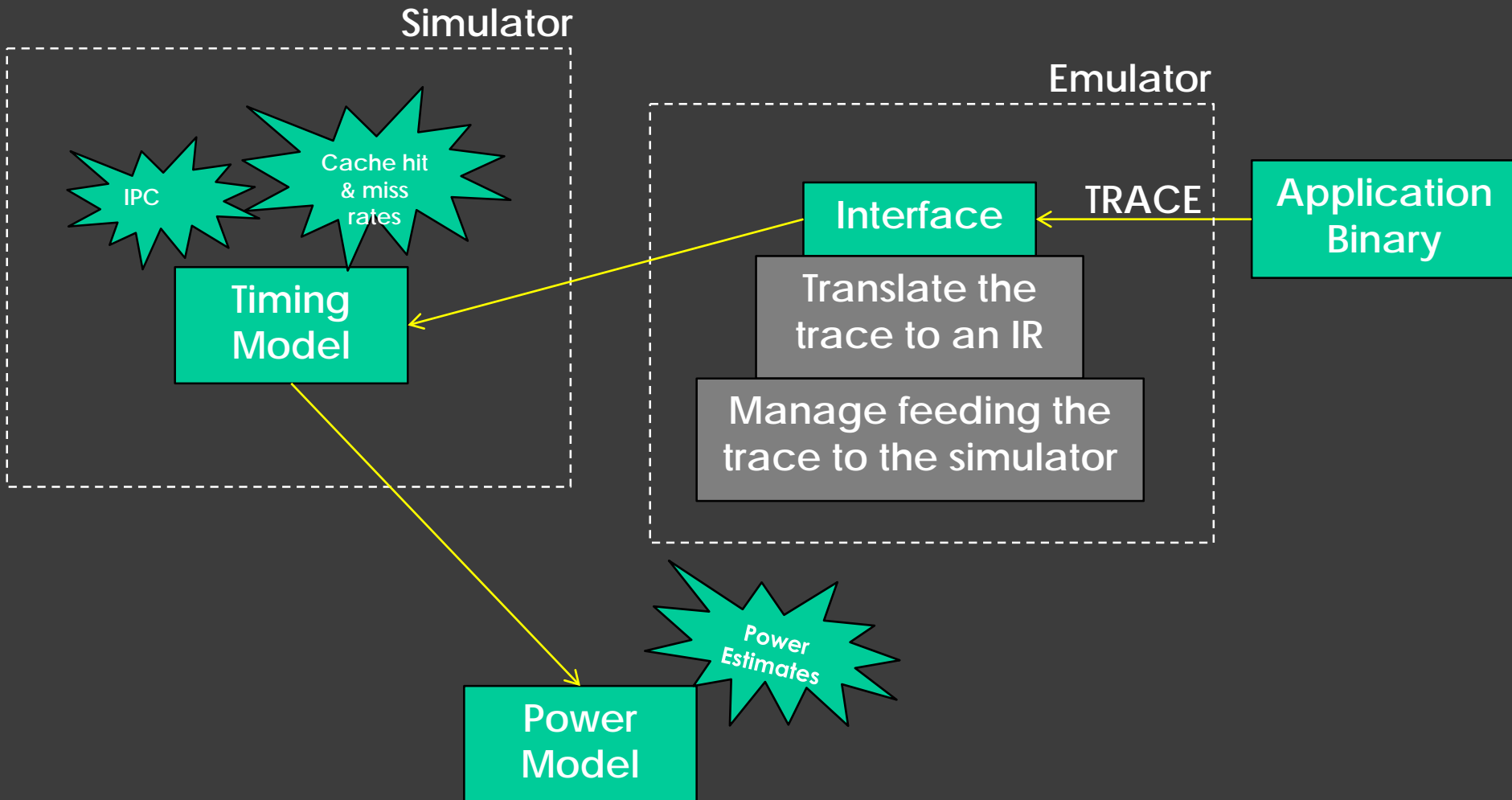
- More computational capability with each new GPU
 - Increasing processing elements with each new generation
- Tighter coupling of the CPU and GPU
 - AMD's APUs, HSA
- Mobile / Embedded applications
 - Emphasis on energy efficiency
- Newer processor architectures like Knights Corner
 - Flexibility in architectural description
 - Ease of extension

Available GPGPU Simulators

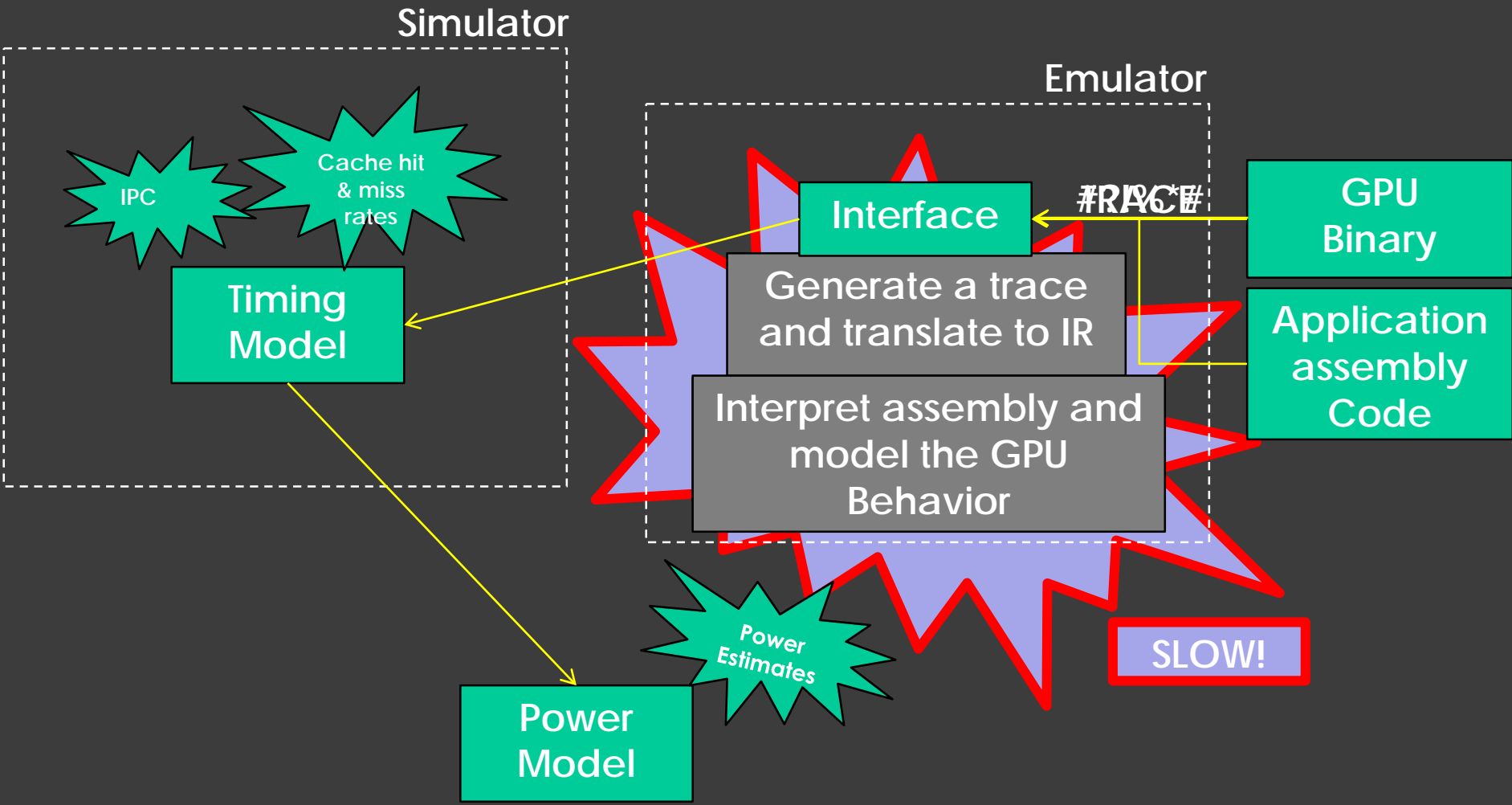
GPGPU Simulators	Key Features
GPGPUSim	Most Popular, Can model Fermi like architectures.
Multi2Sim	Heterogenous simulator, capable of simulating both OpenMP and OpenCL threads.
GPUWattch	Power model for GPGPUs. Now integrated with GPGPUSim
GPUSimPow	Another Power Model, based on GPGPUSim.
Ocelot	Dynamic JIT compilation framework translating PTX to run on several backends

SLOW

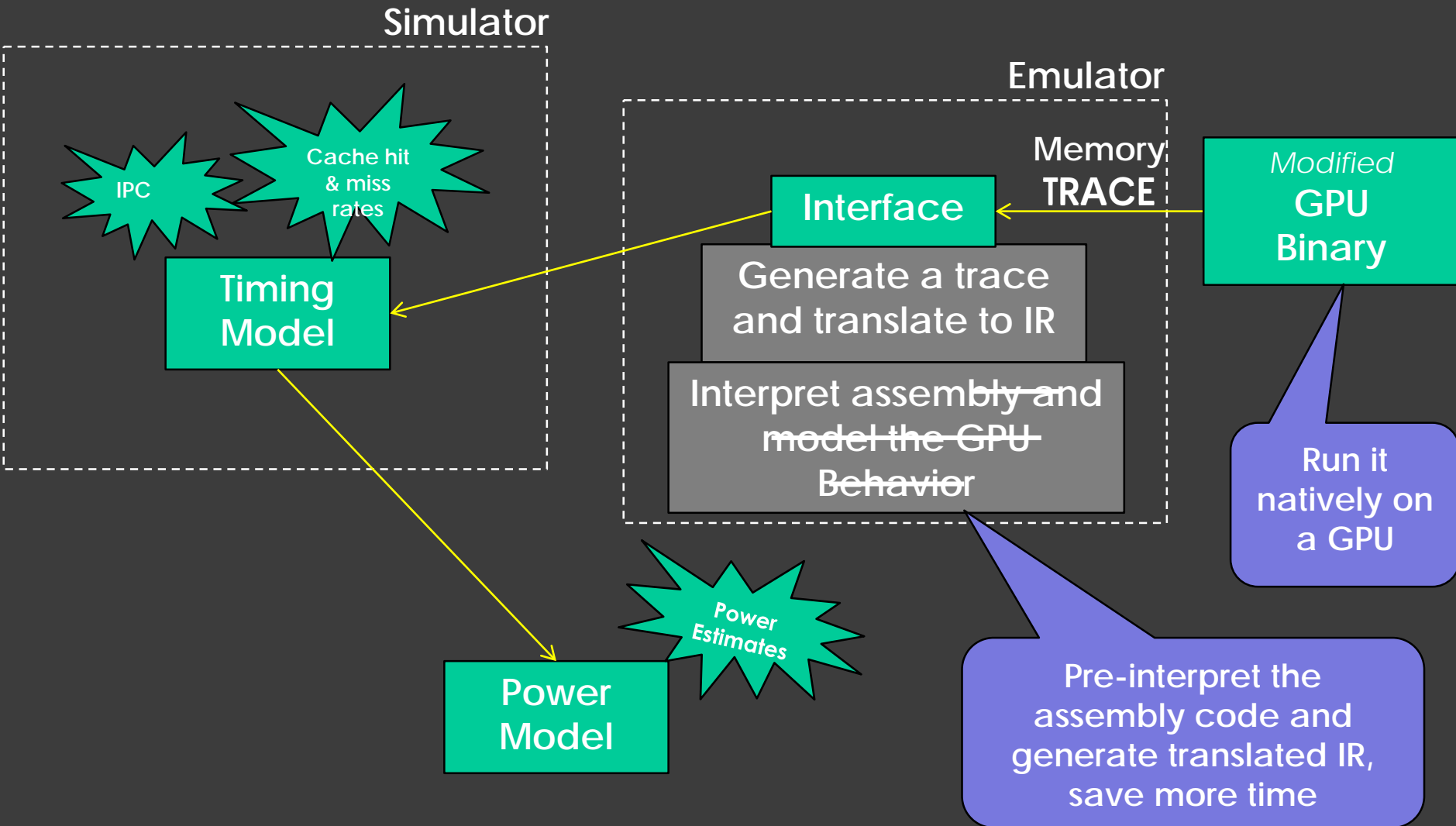
Generic Simulators



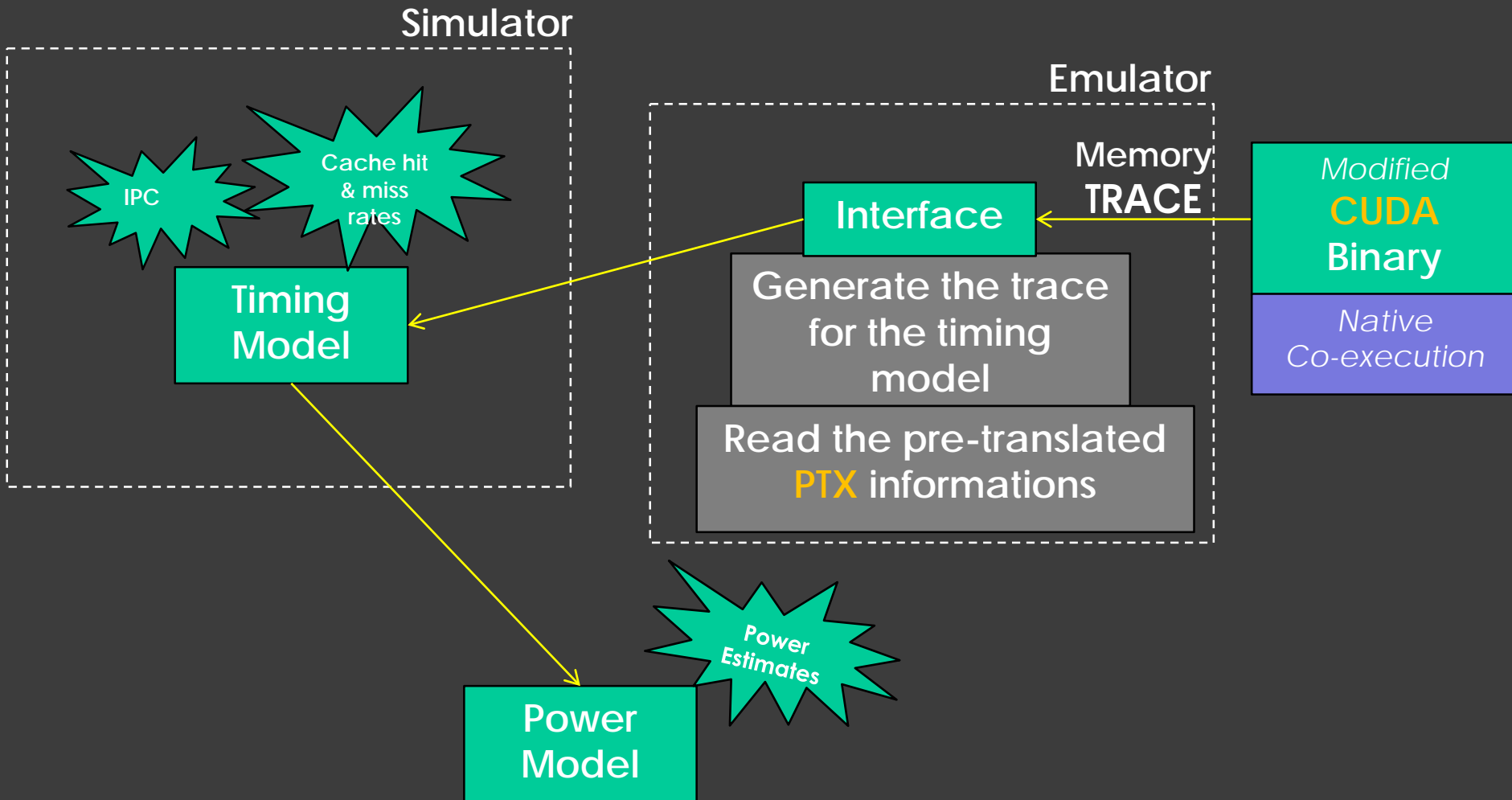
Simulating GPGPUs



How can we make it faster?



Simulating GPGPUs with ESESC



Creating modified binaries

- Purpose
 - Avoid mock GPU execution of the application by the emulator (needed for memory addresses)
 - Generate a trace with the memory addresses, per thread.
 - Exploit the computational power of the GPGPU, to speed up simulation.
- Original application behavior should remain unchanged

Creating modified binaries

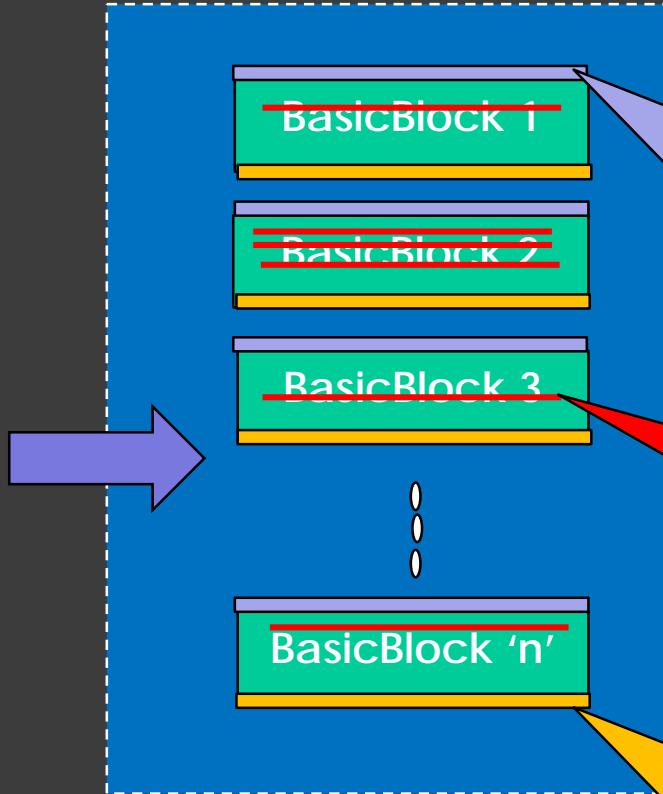
- Challenges

- How can we effectively return the memory addresses per thread?
- How can we convey the execution path of different threads?
(threads can diverge)
- How can we pass the control back and forth between the CPU and the GPU?

Creating modified binaries

"Contaminated" PTX code

CUDA
Application
Assembly
(PTX code)



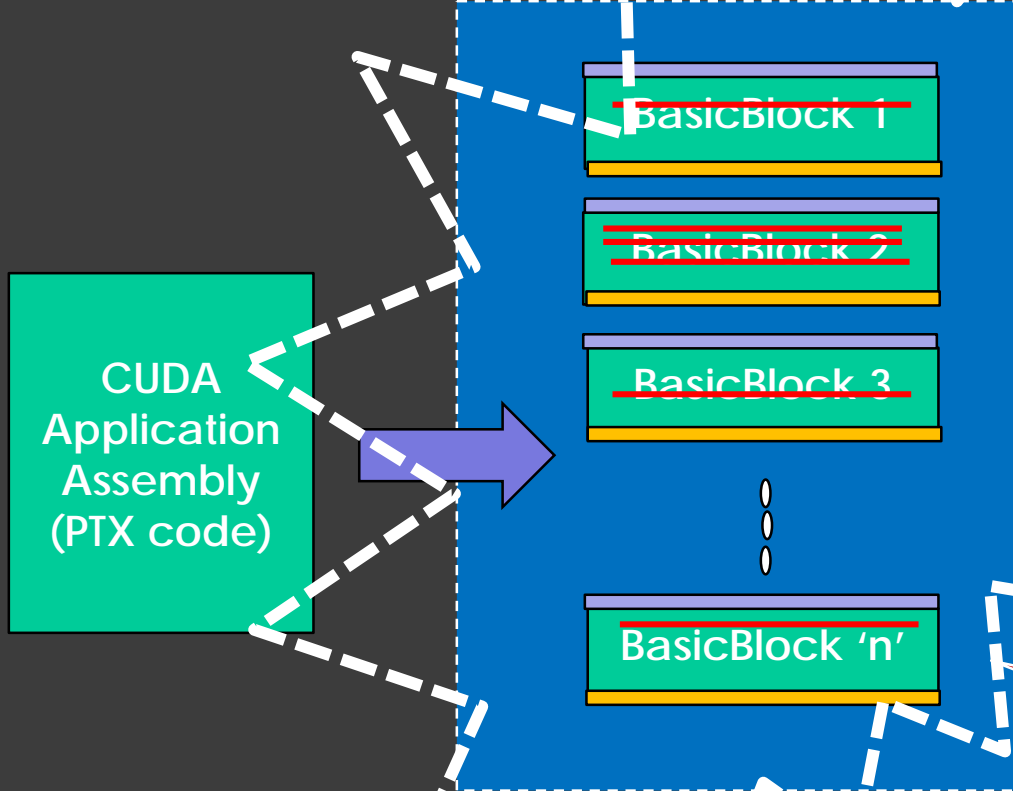
- 1. Load the Live In data (Restore State)
- 2. Save the current BBID

- 1. Save the memory address after each Mem operation

- 1. Save the Live out data (Save State)
- 2. Save the next BBID
- 3. Return control back to the CPU (exit)

Creating modified binaries

"Contaminated" PTX code



Use this "Contaminated" PTX code to create the **modified** application binary.

Contaminated PTX

```
Applications Places System [?] [?] [?] [?]
```

```
bfs.ptx (~/projs/esesc/misc/instdoctor) - VIM
```

```
/bin/bash
```

```
bfs.ptx (~/projs/esesc/misc/instdoctor) - VIM
```

```
69 .param .u32 __cudaparm__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_d_shared_fp64,
```

```
70 .param .u32 __cudaparm__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_d_shared_addr)
```

```
71 {
```

```
72 .reg .u16 %rh<5>;
```

```
73 .reg .u32 %r<36>;
```

```
74 .reg .pred %p<7>;
```

```
75 .loc 20 33 0
```

```
76 $LDWbegin__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_:
```

```
77 mov.u16 %rh1, %ctaid.x;
```

```
78 mul.wide.u16 %r1, %rh1, 512;
```

```
79 cvt.u32.u16 %r2, %tid.x;
```

```
80 add.u32 %r3, %r2, %r1;
```

```
81 ld.param.s32 %r4, [__cudaparm__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_no_of_nodes];
```

```
$LDWbegin__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_:
```

```
mov.u16 %rh1, %ctaid.x;
```

```
mul.wide.u16 %r1, %rh1, 512;
```

```
cvt.u32.u16 %r2, %tid.x;
```

```
add.u32 %r3, %r2, %r1;
```

```
ld.param.s32 %r4, [__cudaparm__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_no_of_nodes];
```

```
setp.le.s32 %p1, %r4, %r3;
```

```
@%p1 bra $Lt_0_5122;
```

```
77 ld.global.s32 %r12, [%r11+0];
```

```
78 mov.s32 %r13, %r12;
```

```
79 ld.global.s32 %r14, [%r11+4];
```

```
100 add.s32 %r15, %r14, %r12;
```

```
101 setp.le.s32 %p3, %r15, %r12;
```

```
102 @%p3 bra $Lt_0_5122;
```

```
103 mul.lo.u32 %r16, %r12, 4;
```

```
104 ld.param.u32 %r17, [__cudaparm__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_g_graph_edges];
```

```
105 add.u32 %r18, %r17, %r16;
```

```
106 ld.param.u32 %r19, [__cudaparm__Z7Kernel1P4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S4_S3_g_graph_visited];
```

```
107 $Lt_0_4098:
```

```
108 //<Loop> Loop body line 40, nesting depth: 1, estimated iterations: unknown
```

```
109 .loc 20 42 0
```

```
[/bin/bash] bfs.ptx (~/projs/esesc/...
```

Contaminated PTX

```
$LDWbegin__27KernelIP4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S3_:
// BBT set BB input begin
// Restore previously computed predicates
ld.global.u32 %bbaux, [%bbtp+48];
setp.eq.u32 %p1, %bbaux,1;
// ..
// BBT set BB input end

$Lt_bbt_0_inside:
// Store Current BBID
st.global.u32 [%bbtp+4], 1;
// ..

mov.u16 %rh1, %ctaid.x;
mul.wide.u16 %r1, %rh1, 512;
cvt.u32.u16 %r2, %tid.x;
add.u32 %r3, %r2, %r1;
ld.param.s32 %r4, [__cudaparam__27KernelIP4NodePiPbS2_S2_S1_iPjS3_PyS3_S4_S3_S4_S3_S3_no_of_nodes];
setp.le.s32 %p1, %r4, %r3;
//Store Predicate
@%p1 st.global.u32 [%bbtp+48], 1;
@!%p1 st.global.u32 [%bbtp+48], 0;
//Store Predicate

//Insert instruction to decrement the skip_inst count
sub.s32 %skipinstcount,%skipinstcount,6;
setp.lt.s32 %pskip_inst,%skipinstcount, 1;
@%pskip_inst st.global.s32 [%bbtp+8], 0;
@%pinst_skip add.s32 %instskipcount,%instskipcount,6;
@%pskip_inst st.global.s32 [%bbtp+36], %instskipcount;
// BBT set BB skip inst end

$Lt_bbt_0_outside:
// BBT set BB output begin
// Increment Branch Count
add.u32 %scratchreg1, %scratchreg1, 1;
st.global.u32 [%bbtp+40], %scratchreg1;
@%p1 st.global.u32 [%bbtp+0], 8; // $Lt_0_5122
@%p1 st.global.u32 [%bbtp+12], 1; // Branch Taken
// Increment Taken Branch Count
@%p1 add.u32 %scratchreg2, %scratchreg2, 1;
@%p1 st.global.u32 [%bbtp+44], %scratchreg2;
@!%p1 st.global.u32 [%bbtp+0], 2; // $Lt_bbt_1
@!%p1 st.global.u32 [%bbtp+12], 0; // Branch not Taken
st.global.u32 [%bbcp32+0], %r11;
st.global.u32 [%bbcp32+4], %r13;
st.global.u32 [%bbcp32+8], %r15;
st.global.u32 [%bbcp32+12], %r3;
@%pskip_inst exit;
@%p1 bra $Lt_bbt_7_inside; // $Lt_0_5122
@!%p1 bra $Lt_bbt_1_inside; // $Lt_bbt_1
// BBT set BB output end
```

1. Load the Live In data (Restore State)
2. Save the current BBID

1. Save the Live out data (Save State)
2. Save the next BBID
3. Return control back to the CPU (exit)

Pre-translated *.info file

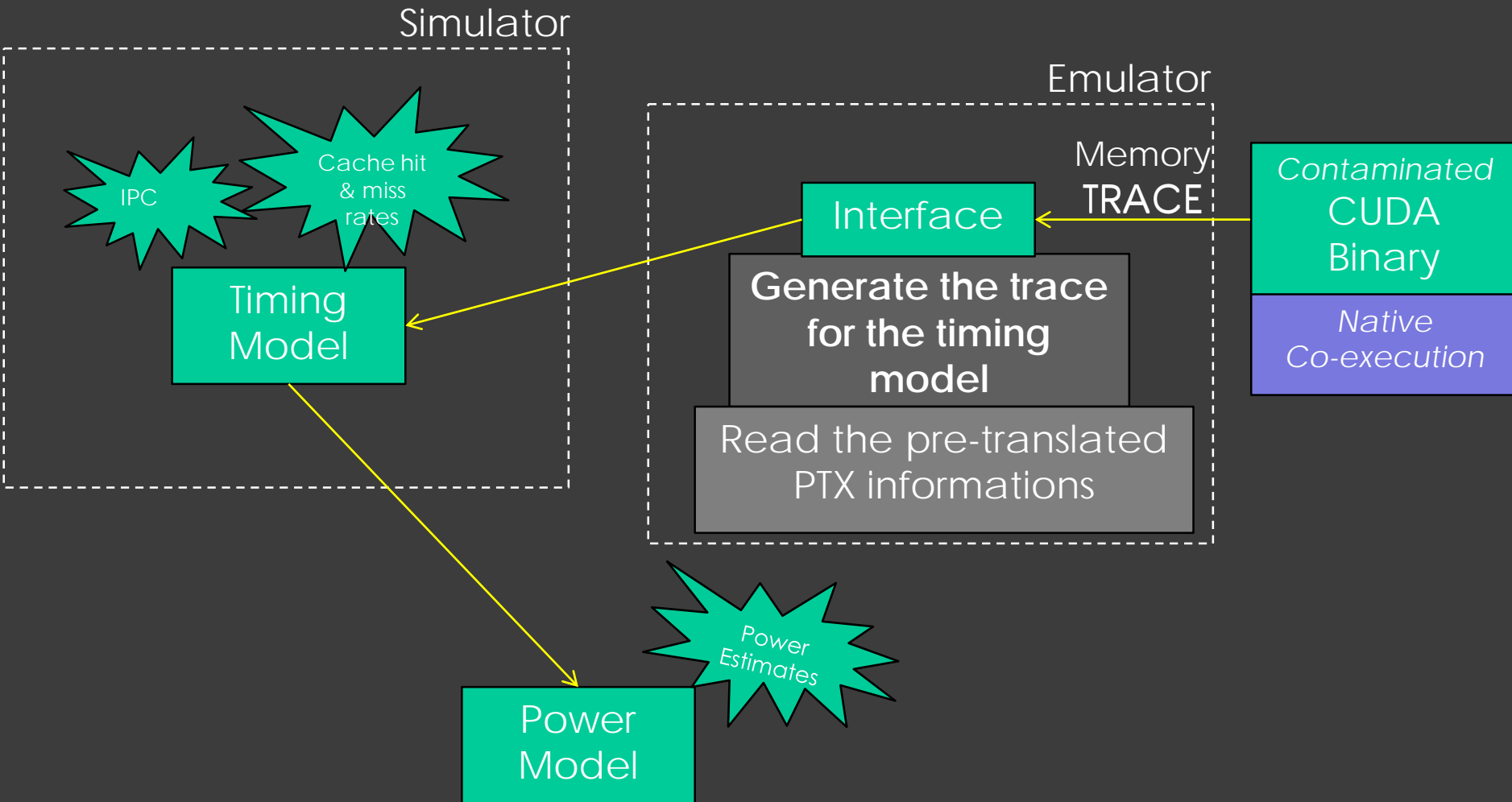
```
1 [KERNEL]
2 Name= _Z7Kernel1P4NodePiPb52_S2_S1_iPj53_Py53_S4_S3_S4_S3_S4_S3_
3 STARTPC=1073741824
4 NUMBER_BB=8
5 DFL_REG32= 8
6 DFL_REG64= 0
7 DFL_REGFP= 0
8 DFL_REGFP64= 0
9 SM_REG32=0
10 SM_REG64=0
11 SM_REGFP=0
12 SM_REGFP64=0
13 SM_ADDR=0
14 PREDICATES=5
15 TRACESIZE=22
16
17 [BB]
18 ID=1
19 Divergent=1
20 Reconv_BB=8
21 Number_insts=7
22 BARRIER=0
23 1073741824,iAALU,0,0,0,ctaid.x,,,rh1
24 1073741828,iCALU_MULT,0,0,0,rh1,512,,r1
25 1073741832,iAALU,0,0,0,tid.x,,,r2
26 1073741836,iAALU,0,0,0,r2,r1,,r3
27 1073741840,iAALU,0,0,0,[_cudaparm__Z7Kernel1P4NodePiPb52_S2_S1_iPj53_Py53_S4_S3_S4_S3_S4_S3_no_of_nodes],,,r4
28 1073741844,iAALU,0,0,0,r4,r3,,p1
29 1073741848,iBALU_BRANCH,3,1,0,,,,sLt_0_5122
30 [END_BB]
31
32 [BB]
33 ID=2
34 Divergent=1
35 Reconv_BB=8
36 Number_insts=6
37 BARRIER=0
38 1073741852,iAALU,0,0,0,[_cudaparm__Z7Kernel1P4NodePiPb52_S2_S1_iPj53_Py53_S4_S3_S4_S3_S4_S3_g_graph_mask],,,r5
39 1073741856,iAALU,0,0,0,r5,r3,,r6
40 1073741860,iLALU_LD,1,0,1,r6,,,r7
41 1073741864,iAALU,0,0,0,0,,,r8
42 1073741868,iAALU,0,0,0,r7,r8,,p2
43 1073741872,iBALU_BRANCH,3,1,0,,,,sLt_0_5122
44 [END_BB]
45
46 [BB]
47 ID=3
48 Divergent=1
49 Reconv_BB=8
50 Number_insts=11
51 BARRIER=0
```

Kernel Name

Trace Statistics

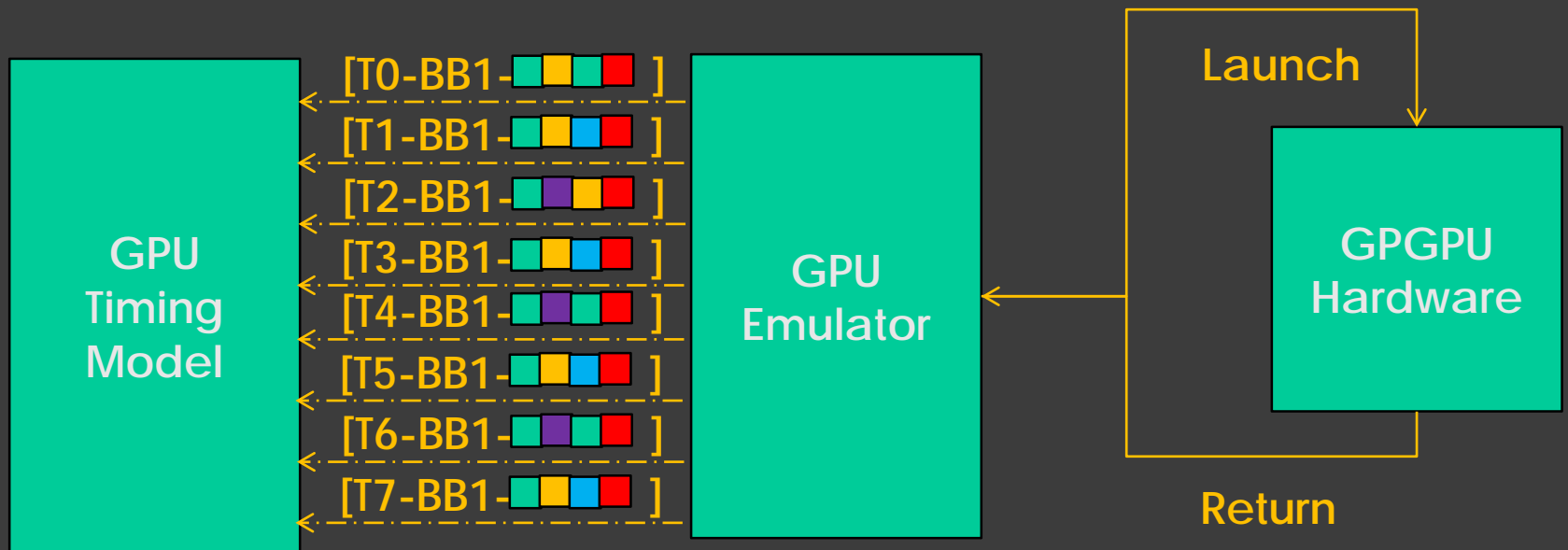
Divergence information.

Simulating a GPGPU











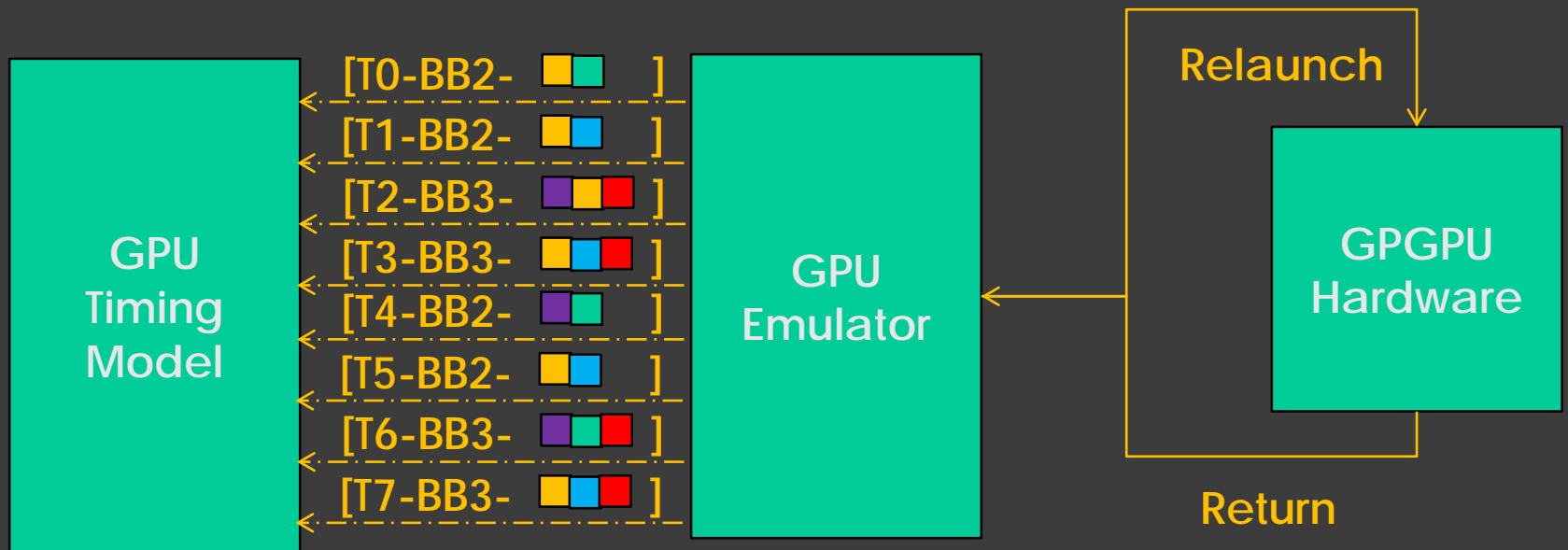
Trace Generation

	T0	T1	T2	T3	T4	T5	T6	T7
Current BBID	1	1	1	1	1	1	1	1
Next BBID	2	2	3	3	2	2	3	3
Memory Addresses								
Done?	0	0	0	0	0	0	0	0











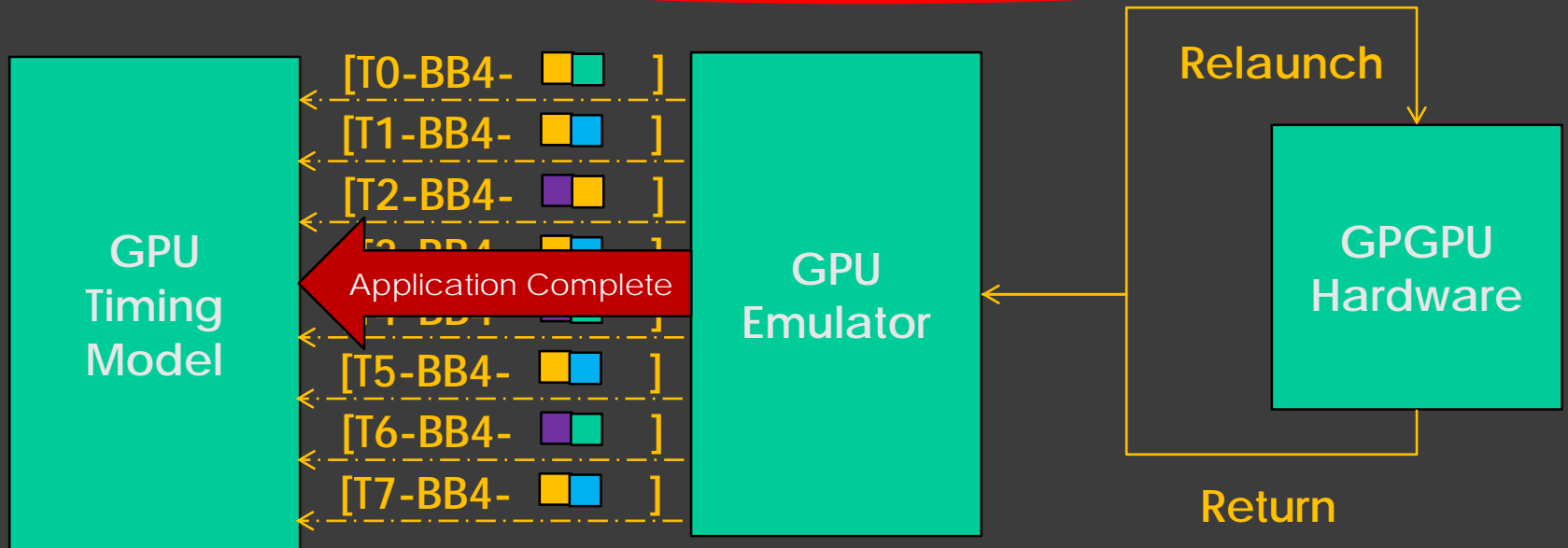
Trace Generation

	T0	T1	T2	T3	T4	T5	T6	T7
Current BBID	2	2	3	3	2	2	3	3
Next BBID	4	4	4	4	4	4	4	4
Memory Addresses								
Done?	0	0	0	0	0	0	0	0

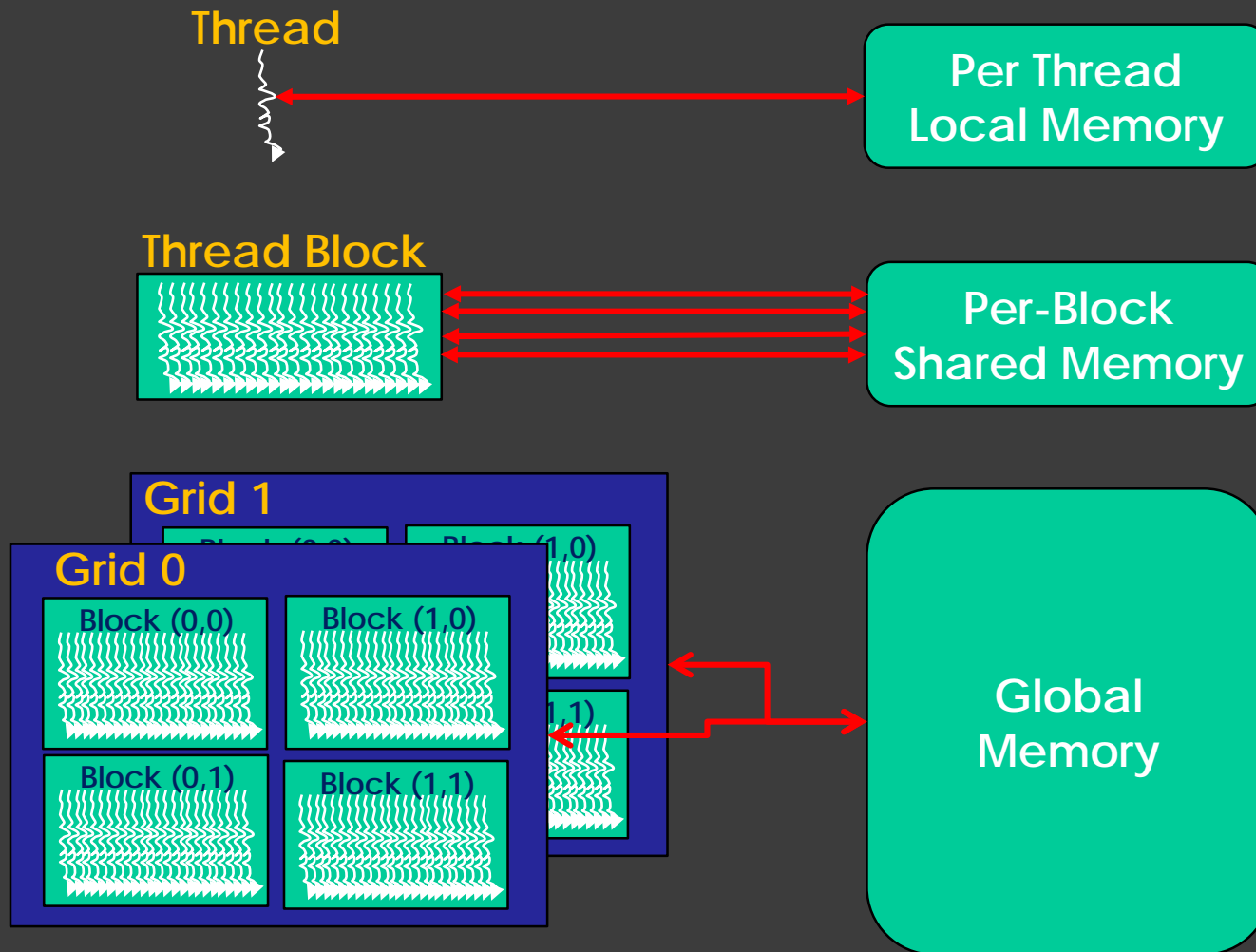


Trace Generation

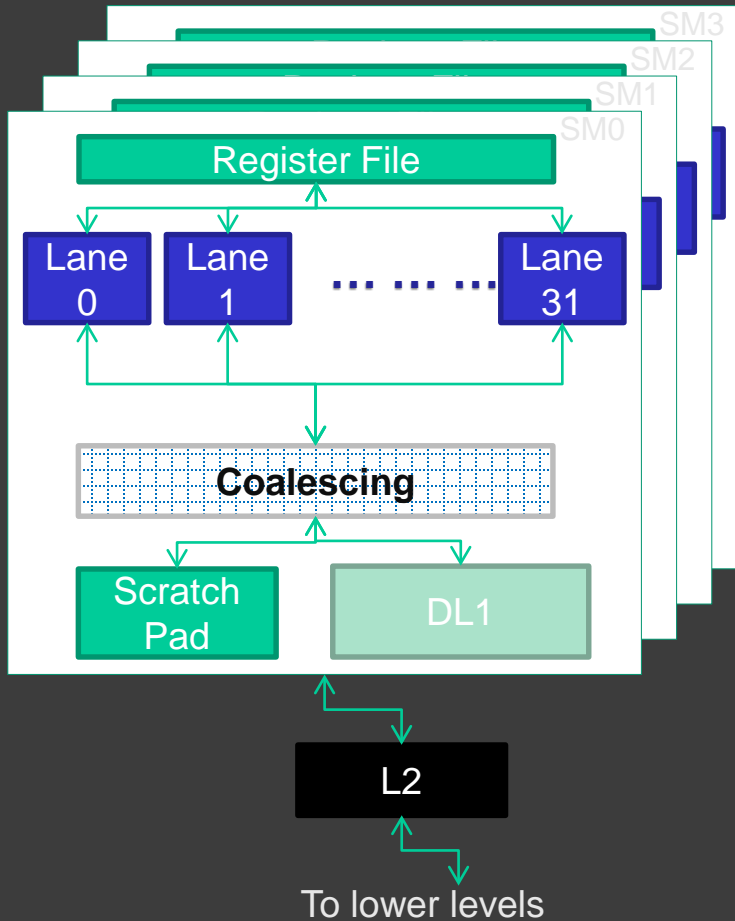
	T0	T1	T2	T3	T4	T5	T6	T7
Current BBID	4	4	4	4	4	4	4	4
Next BBID	0	0	0	0	0	0	0	0
Memory Addresses								
Done?	1	1	1	1	1	1	1	1



A Modern GPGPU

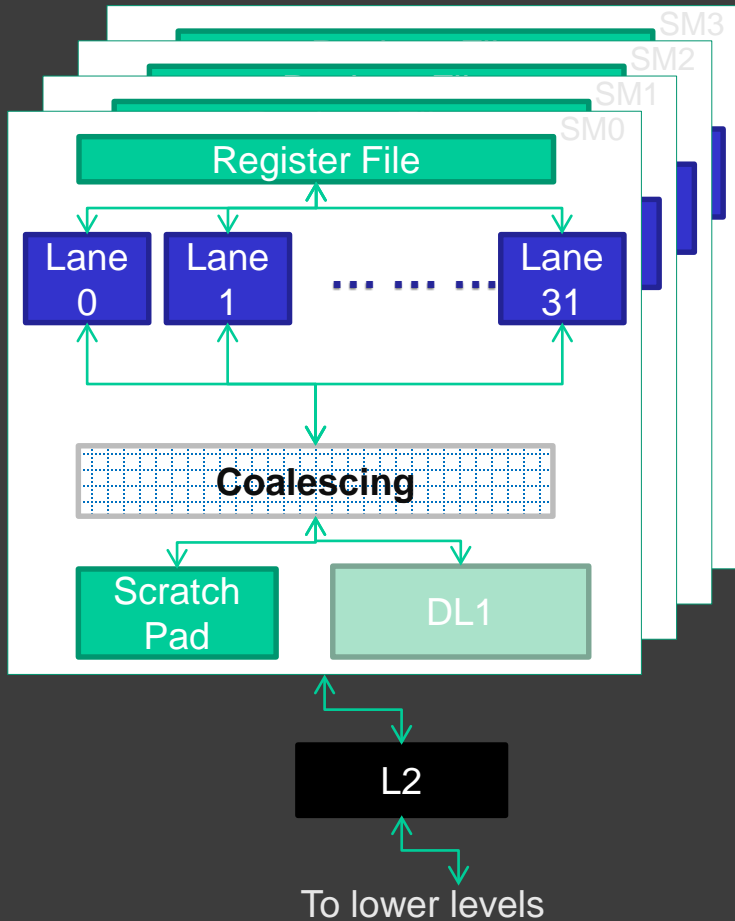


Timing Model



- Each SM is modeled as a group of little cores (lanes)
 - Based on the in-order core modeled in ESESC
 - Each lane can be configured to have the same capabilities as a regular in-order core.
- Graphic specific blocks (rasterizer, clipping) are not modeled

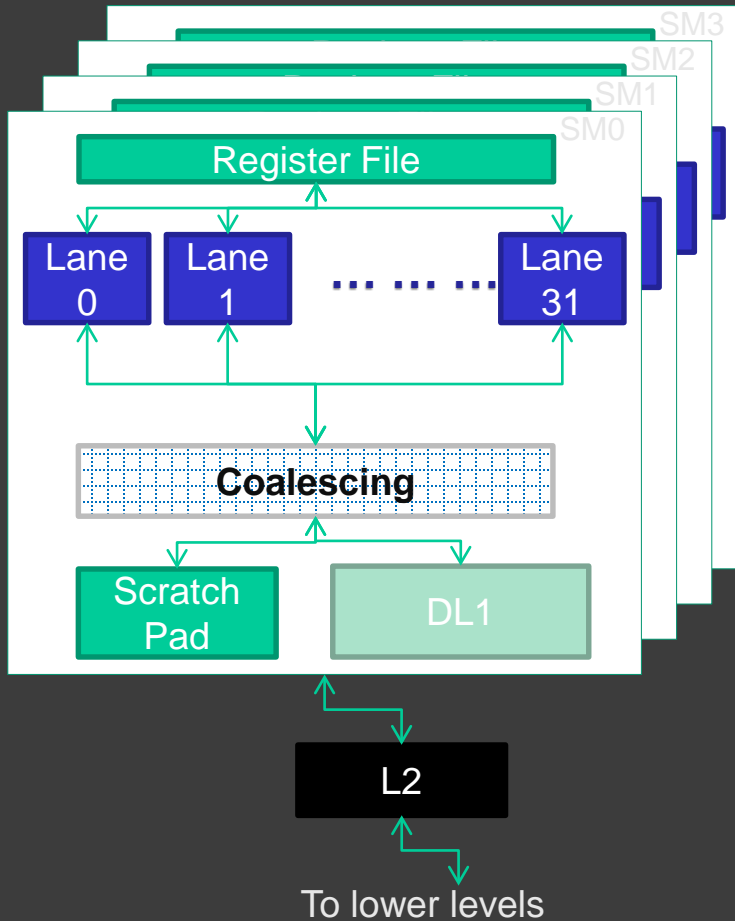
Timing Model



- The trace generator / manager for ESESC models
 - Barriers
 - Execution strategies
 - Divergence mechanisms
 - Serial execution
 - Post Dominator convergence [1]
 - Simultaneous Branch Interleaving [2]

1. Fung, Wilson WL, et al. "**Dynamic warp formation and scheduling for efficient GPU control flow.**" *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2007.
2. Brunie, Nicolas, Sylvain Collange, and Gregory Diamos. "**Simultaneous branch and warp interweaving for sustained GPU performance.**" *ACM SIGARCH Computer Architecture News*. Vol. 40. No. 3. IEEE Computer Society, 2012.

Timing Model



- Memory Hierarchy is defined and used just as for CPU simulations
 - Extensions to indicate if an address is a shared or global address
 - Extensions to indicate which thread or warp a memory address belongs

Software architecture



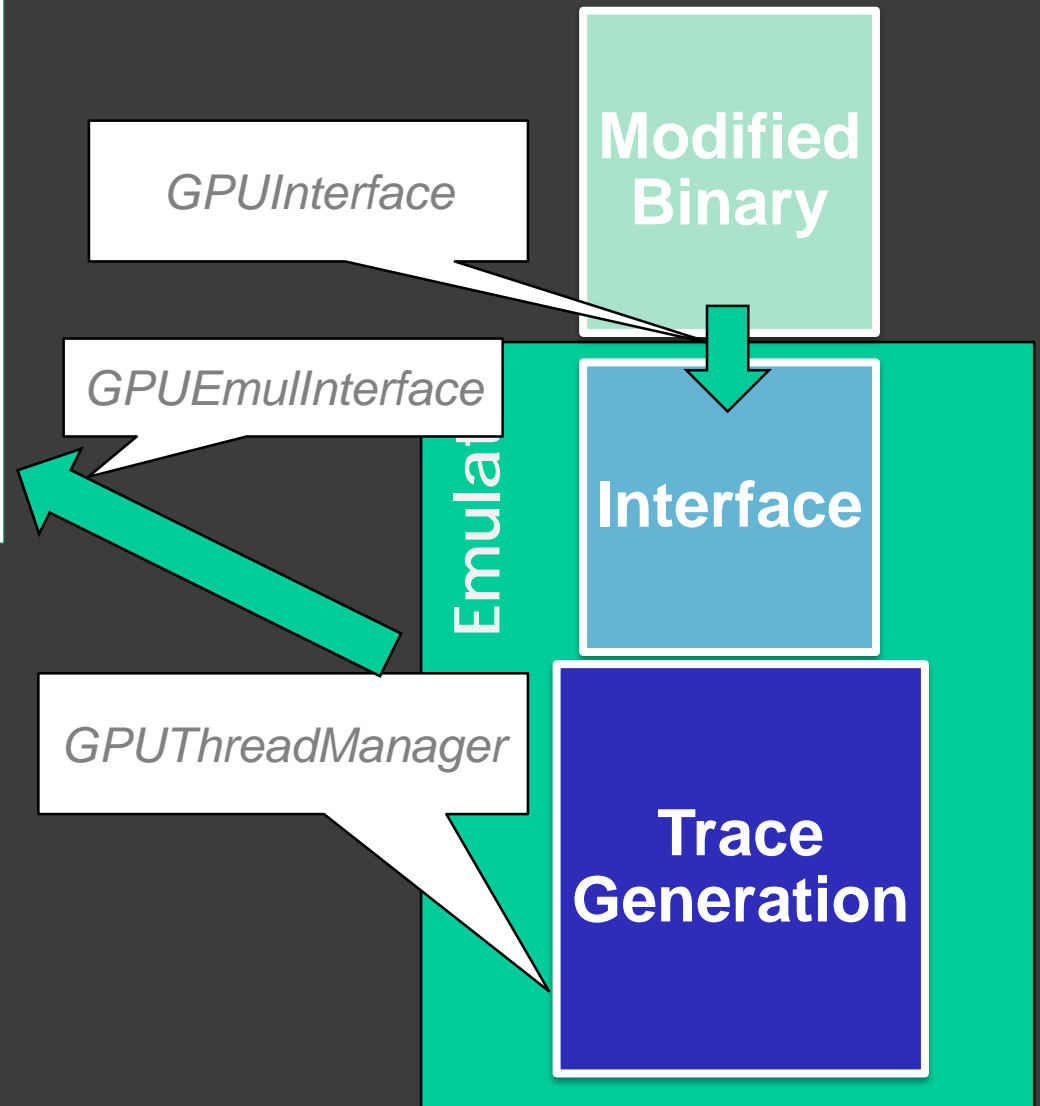
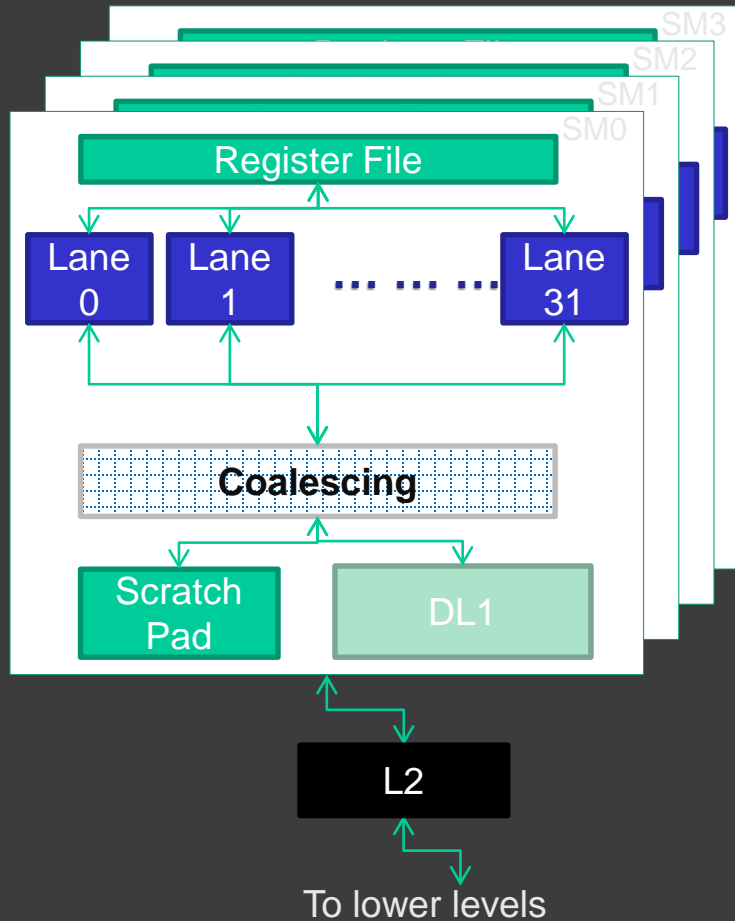
- *InstDoctor* to contaminate PTX
- Custom compilation flow using NVCC

- *GPUInterface*
- Modifications to QEMU

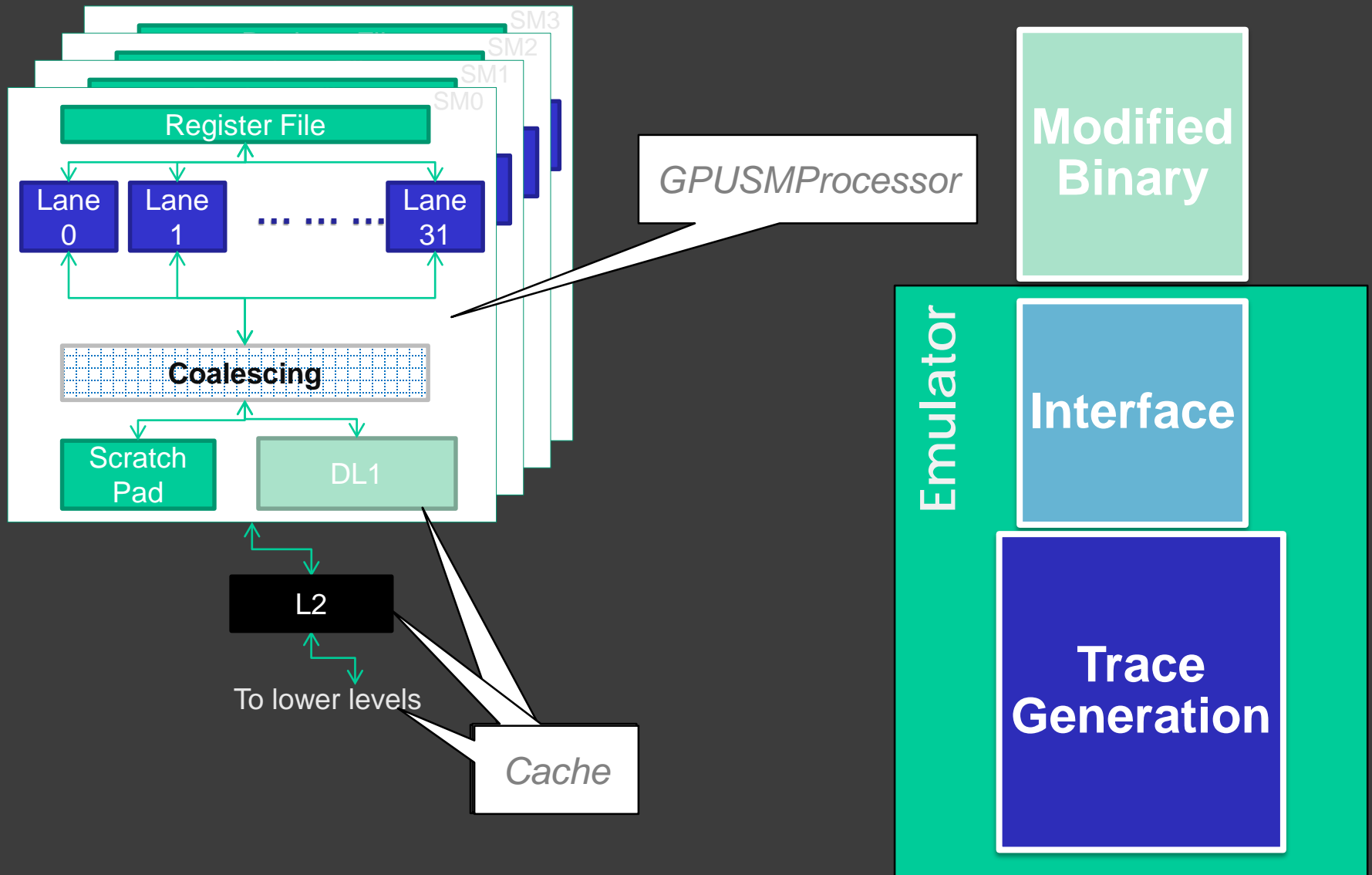
- *GPUThreadManager*
- *GPUEmulInterface*

- *GPUSMPprocessor*
- *gpu.cpp*
- Existing ESESC infrastructure

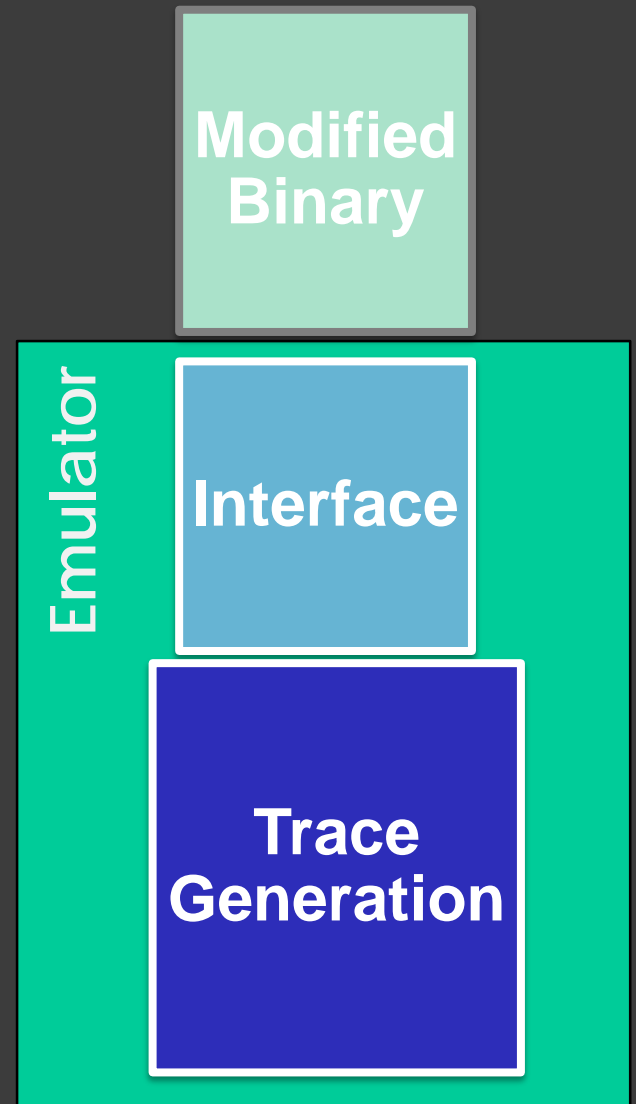
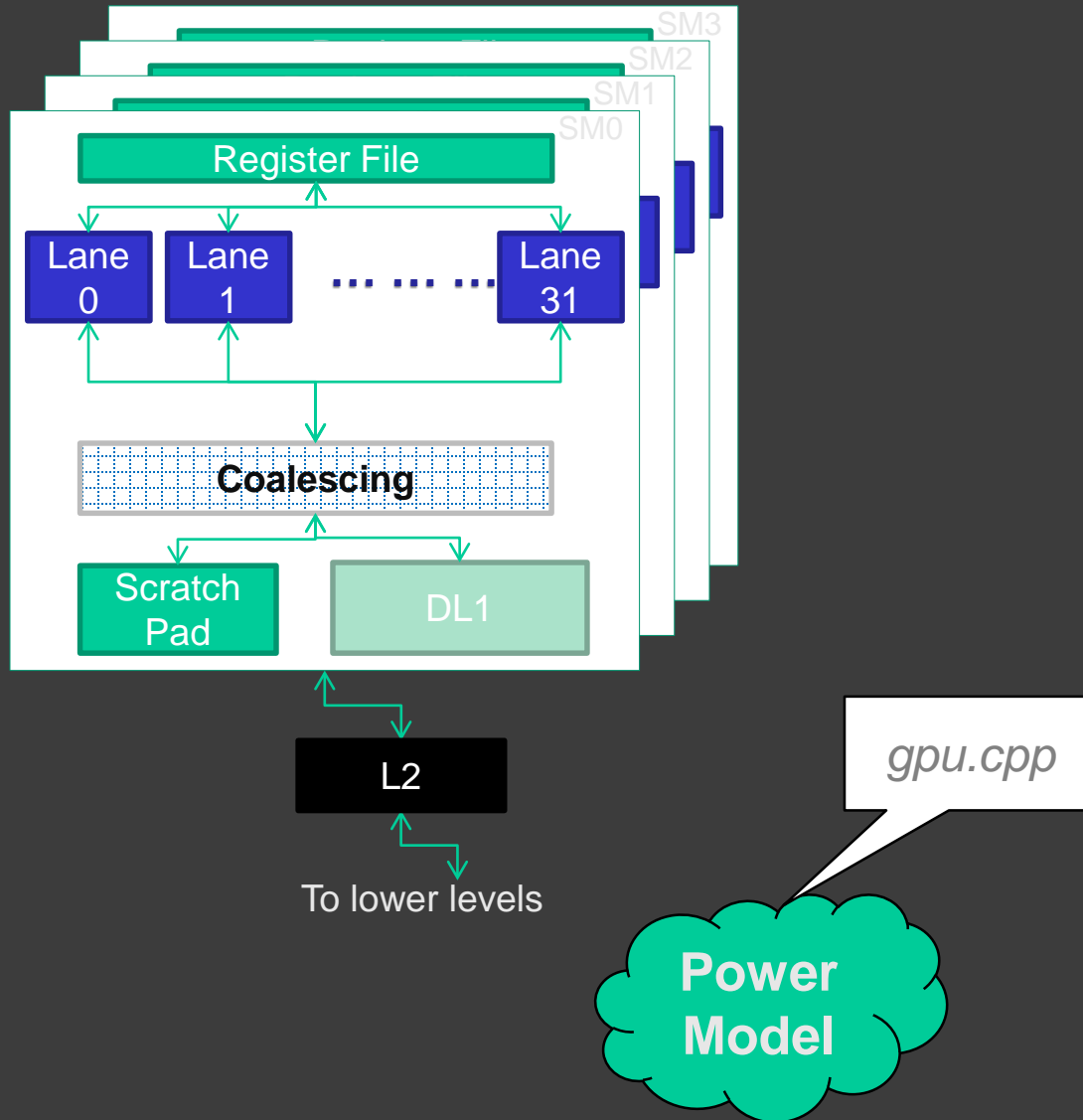
Software architecture



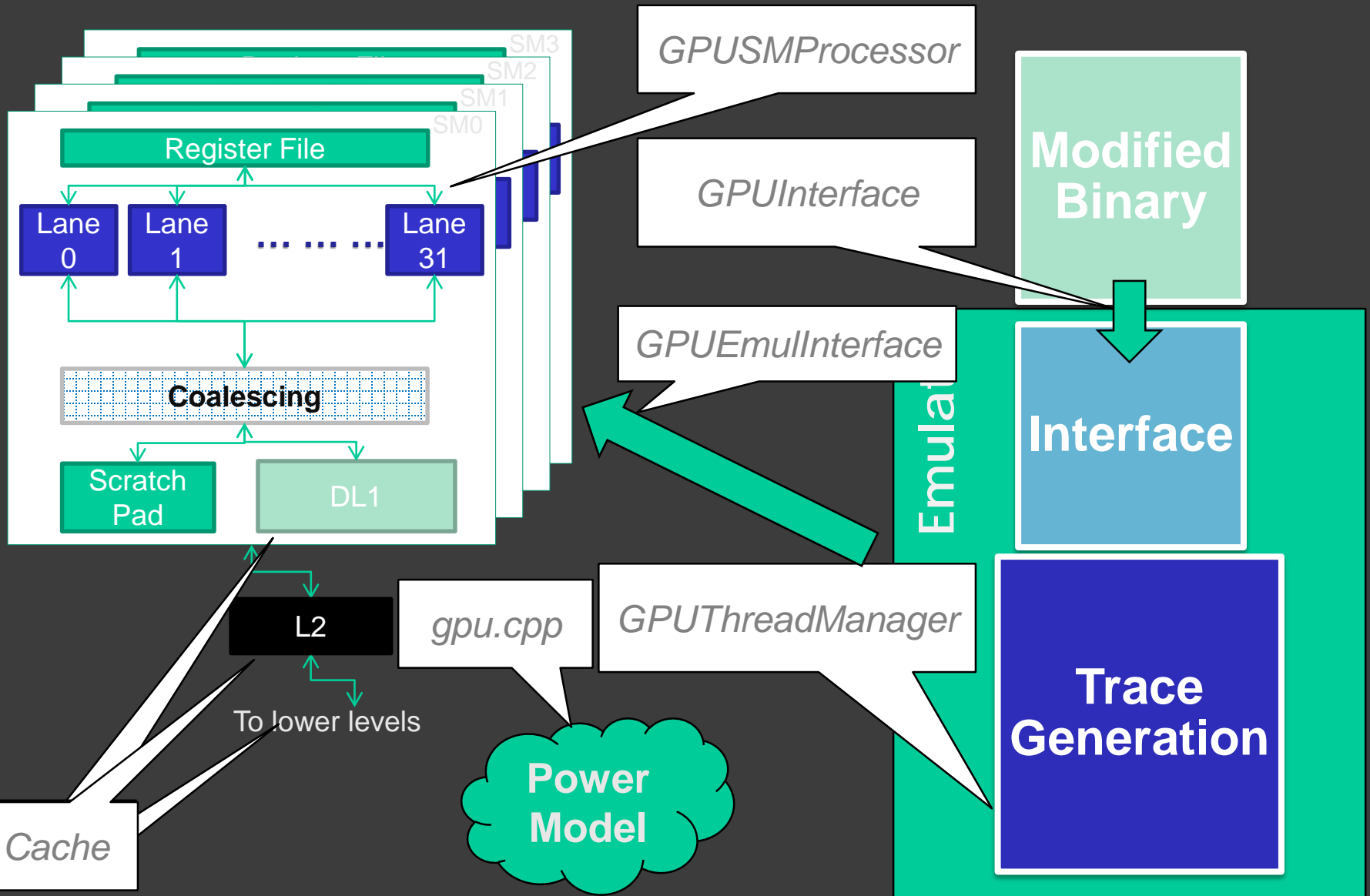
Software architecture



Software architecture



Software architecture



Running a GPGPU application

```
> nvidia-smi
```

```
Tue Jun 10 06:53:20 2014
```

```
+-----+
| NVIDIA-SMI 4.304.51   Driver Version: 304.51           |
+-----+-----+-----+-----+
| GPU   Name           | Bus-Id        Disp. | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+
|    0   GeForce GTX 480 | 0000:01:00.0  N/A  |           N/A        |
| 44%   46C   N/A     N/A /  N/A |  4%   60MB / 1535MB |    N/A      Default  |
+-----+-----+-----+-----+
```

```
+-----+
| Compute processes:                                     GPU Memory |
| GPU       PID  Process name                          Usage         |
+-----+-----+-----+-----+
|    0           Not Supported                          |
+-----+-----+-----+-----+
```

Running a GPGPU application

- Step 1 : Creating a contaminated binary
 - Code cleanup in progress, detailed instructions will be made available soon after.
 - A few contaminated binaries will be provided for now.

Running a GPGPU application

- Step 2: Compiling esesc.
 - Need two additional flags
 - Enable 32 bit mode
 - Enable GPU mode (link with CUDA libraries)
 - Command to build in Release Mode

```
> cmake
  -DCMAKE_HOST_ARCH=i386
  -DENABLE_CUDA=1
  ~/projs/esesc
```

Running a GPGPU application

• Step 3 : Configure esesc.conf

```
# Select simulated core type. Defined in simu.conf
coreType    = 'tradCORE'
#coreType   = 'scoreCORE'
SMcoreType  = 'gpuCORE' ← NOTE! New coretype for GPGPU
```

```
# Sampling mode
samplerSel  = "TASS"
gpusampler  = "GPUSpacialMode" ← NOTE! Sampling?
```

```
# Set the correct number of processors
cpuemul[0]  = 'QEMUSectionCPU'
cpuemul[1:4] = 'QEMUSectionGPU' ← NOTE! Section where additional GPU parameters are specified

cpusimu[0]  = "$(coreType)"
cpusimu[1:4] = "$(SMcoreType)" ← NOTE! Number of SMs

SP_PER_SM  = 32 ← NOTE! Number of Lanes
```

Running a GPGPU application

• Step 3 : Configure esesc.conf

```
benchName = "-s 8192000 kernels/bfs kernels/graph4096.txt"  
infofile  = "kernels/bfs.info"  
reportFile = 'gpu_bfs' ← NOTE! Pre-translated PTX  
MAXTHREADS = 1024
```

```
enablePower = true
```

```
[GPUSpacialMode]  
type = "GPUSpacial" ← NOTE! Special Sampler for GPU  
nMaxThreads = $(MAXTHREADS) ← NOTE! Selective  
execution of threads  
nInstSkip = 0  
nInstMax = 1e14
```

Sampling, for GPGPUs?

- GPGPU applications are largely homogeneous
- Do we need to execute and simulate all the threads?
- Use “MAXTHREADS” to simulate the first “\$(MAXTHREADS)” threads.
 - The others are executed natively on hardware (for correct execution)
- Extract significant speedup!
 - Need to profile applications to see how much we can skip simulating

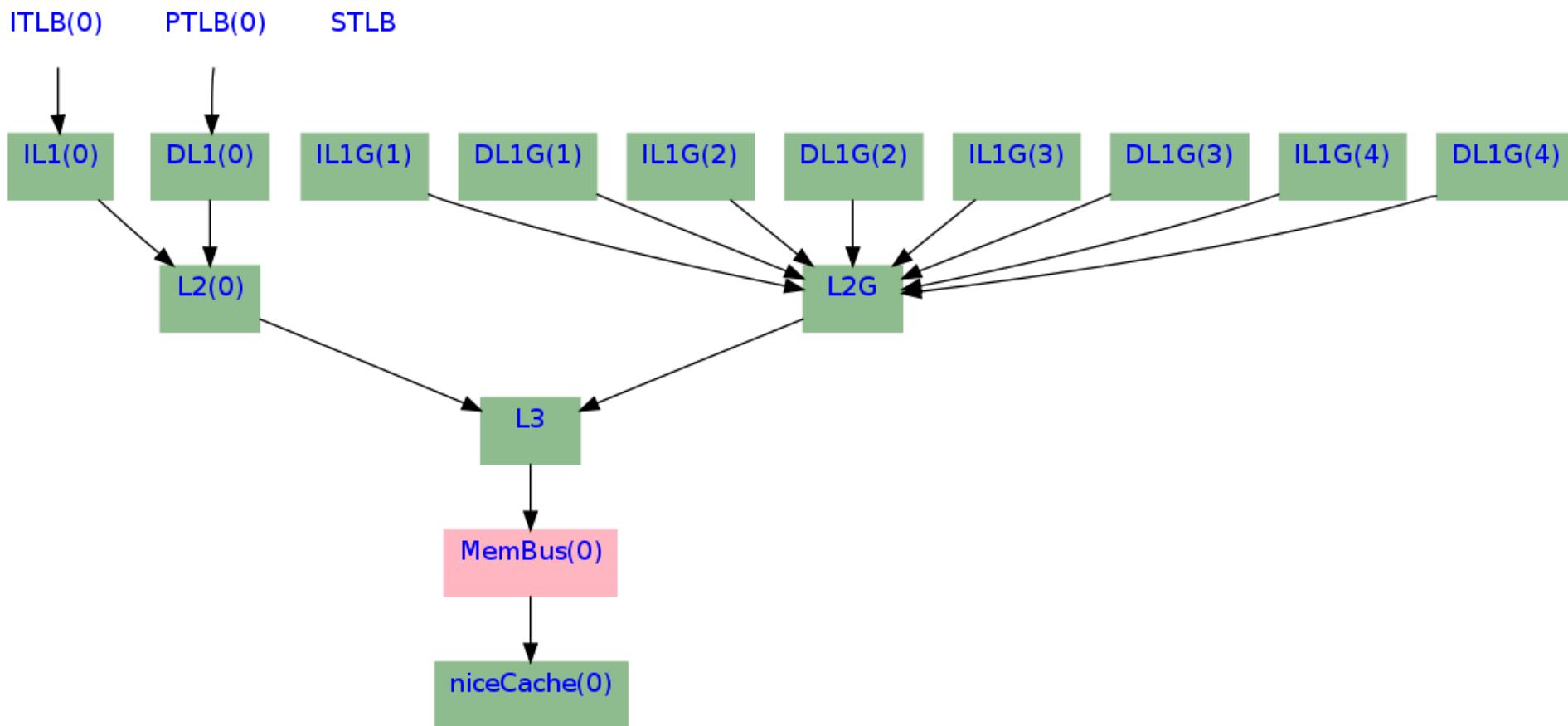
Running a GPGPU application

- Step 4 : Configure simu.conf (if needed)

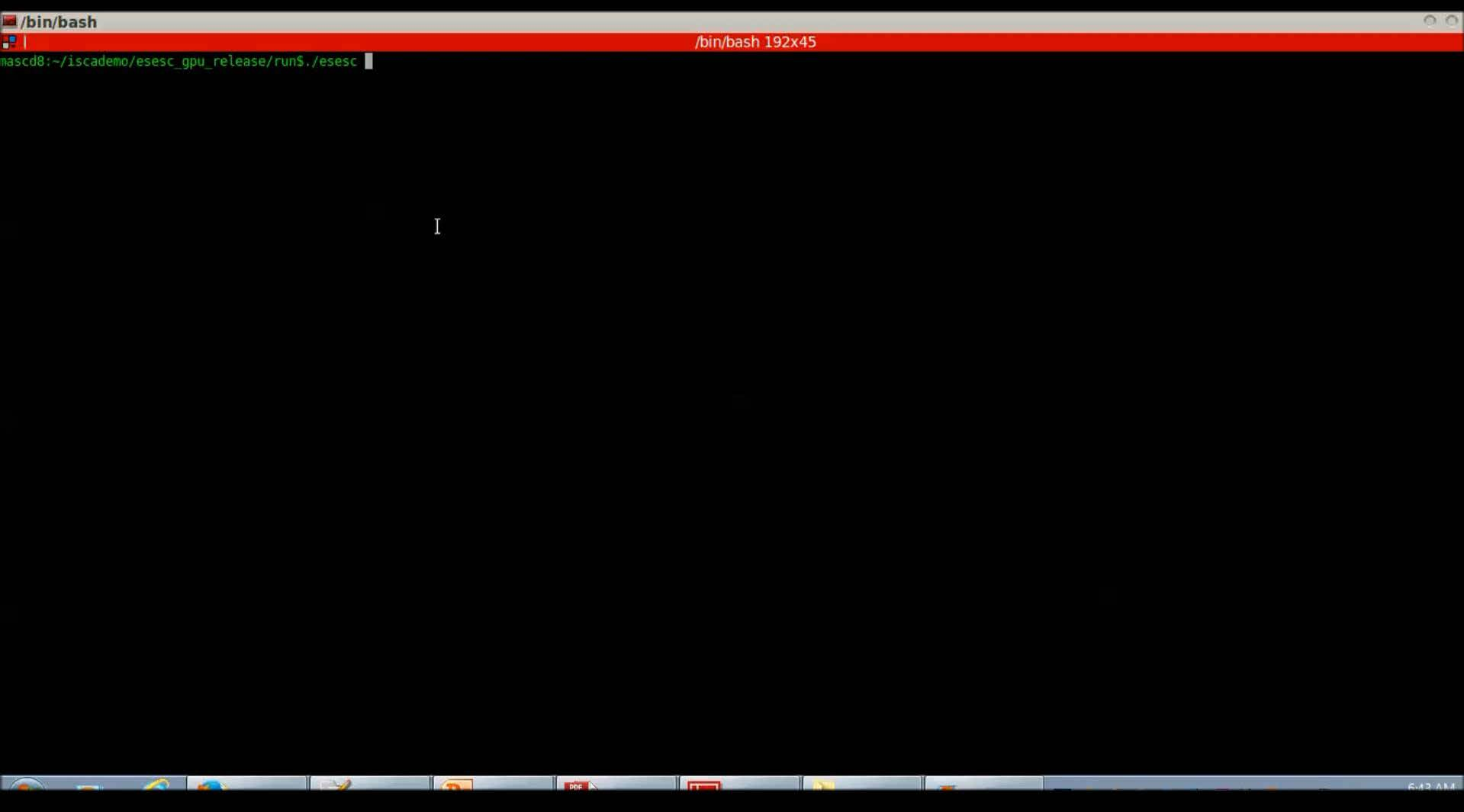
```
[gpuCORE]
sp_per_sm          = $(SP_PER_SM) #needed to instantiate the GPU SM
                                   #Processor
areaFactor         = 2             # Area in relation with alpha264 EV6
issueWrongPath     = false
fetchWidth         = $(SP_PER_SM)
instQueueSize     = $(SP_PER_SM)*2
inorder           = true
throttlingRatio   = 2.0
issueWidth        = $(SP_PER_SM)
retireWidth       = $(SP_PER_SM)
decodeDelay       = 3*2
renameDelay       = 2*2
.
.
.
```

Running a GPGPU application

- Step 4 : Configure simu.conf (if needed)



Running a GPGPU application



A terminal window titled "/bin/bash" with a red title bar. The window shows the prompt "mascd8:~/iscademo/esesc_gpu_release/run\$./esesc" followed by a single character "I". The window is 192x45 pixels. The desktop background is black, and a taskbar is visible at the bottom with various application icons and a system clock showing 6:43 AM.

Sample Report

```

/bin/bash 223x53
Sampler 1 (Procs 1 2 3 4) (24.441 million Rabbit, 0.825 million Timing ,25.266 million Total PTX Instructions)
Rabbit Warmup Detail Timing Total KIPS
KIPS 158031 N/A N/A 1134 28631 17.5% 0.0% 0.0% 82.5% ; Sim Time (s) 0.113 Exe 0.035 ms Sim (1700MHz)
Inst 96.7% 0.0% 0.0% 3.3% : Approx Total Time 0.000 ms Sim (1700MHz)
-----
Proc : Avg.Time : BPTYPE : Total : RAS : BPred : BTB : BTAC
1 : 596552.756 : nottakenenhanced : 58.40% : ( 0.00% of 0.00%) : 58.40% : ( 99.96% of 9.24%) : 0.00%
2 : 557951.202 : nottakenenhanced : 58.70% : ( 0.00% of 0.00%) : 58.71% : ( 99.96% of 9.53%) : 0.00%
3 : 553277.710 : nottakenenhanced : 58.21% : ( 0.00% of 0.00%) : 58.21% : ( 99.96% of 9.34%) : 0.00%
4 : 574027.803 : nottakenenhanced : 58.39% : ( 0.00% of 0.00%) : 58.39% : ( 99.96% of 9.61%) : 0.00%
-----
Proc : rawInst : nCommit : nInst : AALU : BALU : CALU : LALU : SALU : LD Fwd : Replay : Worst Unit (clk)
1 : 205292 : 205117 : 205118 : 68.39% : 13.14% : 6.44% : 9.84% : 2.19% : 0.00% : N/A : GUNIT_ALU 0.01
2 : 207831 : 207656 : 207657 : 68.26% : 13.07% : 6.45% : 9.95% : 2.26% : 0.00% : N/A : GUNIT_ALU 0.01
3 : 204967 : 204792 : 204793 : 68.43% : 13.19% : 6.40% : 9.82% : 2.15% : 0.00% : N/A : GUNIT_ALU 0.00
4 : 206928 : 206753 : 206754 : 68.34% : 13.16% : 6.40% : 9.91% : 2.19% : 0.00% : N/A : GUNIT_ALU 0.01
-----
Proc IPC uIPC Active Cycles Busy LDQ STQ IWin ROB Regs IO maxBr MisBr Div Br4Clk brDelay
1 14.29 3.55 0.00 57725 11.1 0.0 0.0 0.0 7.8 0.0 0.0 0.0 0.0 361832.9 0.0 0.0
2 00.00 3.52 0.00 59034 11.0 0.0 0.0 0.0 8.6 0.0 0.0 0.0 0.0 330826.9 0.0 0.0
3 00.00 3.57 0.00 57392 11.2 0.0 0.0 0.0 8.3 0.0 0.0 0.0 0.0 339851.7 0.0 0.0
4 00.00 3.63 0.00 57000 11.3 0.0 0.0 0.0 6.6 0.0 0.0 0.0 0.0 355872.1 0.0 0.0
-----
Cache Occ AvgMemLat MemAccesses MissRate ( RD , WR, BUS) Dyn_Pow (mW) Lkg_Pow (mW)
IL1G(1) 0.0 4.2 43085 0.03% (100.0%, 0.0%, 0.0%)
IL1G(2) 0.0 4.2 43476 0.03% (100.0%, 0.0%, 0.0%)
IL1G(3) 0.0 4.2 43304 0.03% (100.0%, 0.0%, 0.0%)
IL1G(4) 0.0 4.2 43819 0.03% (100.0%, 0.0%, 0.0%)
-----
DL1G(1) 0.0 47.4 24683 20.48% ( 79.0%, 36.5%, 0.0%)
DL1G(2) 0.0 46.3 25369 20.95% ( 79.1%, 33.5%, 0.0%)
DL1G(3) 0.0 46.7 24521 20.04% ( 79.8%, 36.8%, 0.0%)
DL1G(4) 0.0 51.3 25023 20.33% ( 79.6%, 35.8%, 0.0%)
-----
L2G(0) 0.0 94.3 22819 18.38% ( 91.9%, 48.8%, 0.0%)
niceCache(0) 0.0 0.0 0 0.00% (100.0%, 0.0%, 0.0%) 0 0
-----
CPU Power Metrics: (Dynamic Power,Leakage Power)
Proc | RF (mW) | ROB (mW) | fetch (mW) | EXE (mW) | RNU (mW) | LSU (mW) | Total (W)
-----
0 | 0 , 0 | 0 , 0 | 0 , 0 | 0 , 0 | 0 , 0 | 0 , 0 | 0.00 , 0.00
-----
GPU Power Metrics: (Dynamic Power,Leakage Power)
SMID | RF (mW) | ExeUs (mW) | IL1G (mW) | DL1G (mW) | DTLBG (mW) | ScrtchP(mW) | Total (W)
-----
1 | 636 , 0 | 452 , 0 | 63 , 0 | 3579 , 0 | 23 , 0 | 11 , 0 | 4.76 , 0.00
2 | 646 , 0 | 468 , 0 | 63 , 0 | 3648 , 0 | 23 , 0 | 11 , 0 | 4.86 , 0.00
3 | 636 , 0 | 453 , 0 | 63 , 0 | 3518 , 0 | 22 , 0 | 11 , 0 | 4.70 , 0.00
4 | 639 , 0 | 456 , 0 | 64 , 0 | 3608 , 0 | 23 , 0 | 11 , 0 | 4.80 , 0.00
-----
L2G Power = 5 (Dyn) , 0 (Lkg)
L3 Power = 0 (Dyn) , 0 (Lkg)
Total GPU Power = 19.13 (Dyn) 0.00 (Lkg)

```

Roadmap

- Still in an early stage.
 - Code cleanup
 - Update the compilation flow to more recent versions of CUDA
 - Add support for newer features released with newer CUDA versions.
- Validation
 - Performance
 - Power

Summary

- ESESC provides a fully **customizable platform** to model GPGPUs
- One of the key differentiators is the enormous speedups we achieve with techniques like **native co-execution** and **selective thread execution**
- Integrated **timing** and **power model**
- Very early stages, but expect to release a stable version in the coming months.

Questions?

ESESC Mailing List

esesc@googlegroups.com

GPU Specific questions

alamelu@soe.ucsc.edu

Acknowledgements

- Dr José Luis Briz Velasco
 - Profesor Titular
 - Associate Professor Computer Architecture and Technology Depto. de Informática e Ingeniería de Sistemas (DIIS) Escuela de Ingeniería y Arquitectura - University of Zaragoza (UZ)
 - briz@unizar.es
- Dr Ehsan K. Ardestani
 - ehsanardestani@gmail.com

Backup Slides

Backup 1 : Speedups

GPGPU Simulators

Slowdown compared to Native

GPGPUSim [2013]

90000 (1350s)[1]

Multi2Sim

8700 (functional)
44000 (arch simulation)[1]

Backup 2 : List of available contaminated benchmarks

Benchmark	Benchmark Suite	#Threads
BACKPROP		1048576
BFS		1000000
CFD		97152
HOTSPOT		1893376
KMEANS		495616
LEUKOCYTE		104296

1. John A. Stratton, Christopher Rodrigues, I-Jui Sung, Nady Obeid, Li-Wen Chang, Nasser Anssari, Geng Daniel Liu, Wen-mei W. Hwu **IMPACT Technical Report**, IMPACT-12-01, University of Illinois, at Urbana-Champaign, March 2012
2. Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. **Rodinia: A benchmark suite for heterogeneous computing**. In *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*(IISWC '09). IEEE Computer Society, Washington, DC, USA, 44-54. DOI=10.1109/IISWC.2009.5306797 <http://dx.doi.org/10.1109/IISWC.2009.5306797>