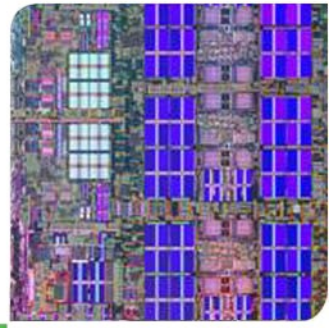
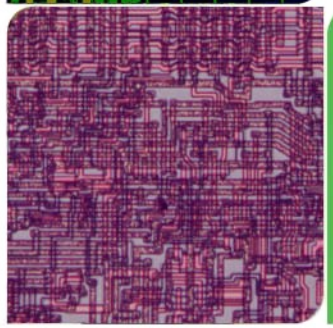
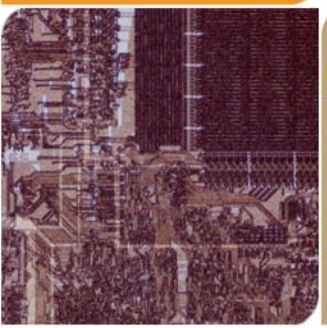





MacSim Tutorial Part-III

Jaekyu Lee

Georgia Tech comparch



Agenda (Part III)

- Source code directory structure
- Trace Generation
- Memory System
- Process Manager
- Data structures
- Knobs and Stats
- Debugging

Georgia Tech comparch



Source Code

Jaekyu Lee

Georgia Tech comparch

MacSim Source Tree

MacSim Top-level

```
bin/        Build output directory
def/        Contain definitions of parameter and events for
statistics
doc/        Documentation (doxygen)
params/     Include sample parameter configuration files
scripts/    Scripts to build MacSim binary
src/        Source files (.cc and .h)
tools/      Tools (x86 trace generator, trace reader, …)
```

```
// These files containing procedure to build and run MacSim and
SConstruct, Sconscript, buildy.py
```

```
// macsim-sst
Makefile.am
macsimComponent.cpp
macsimComponent.h
```

Build Process Demo

Doxygen Documentation

- Detailed MacSim class (members, hierarchy) information
- **Doxygen url:**
  - http://comparch.gatech.edu/hparch/macsim/doxygen/html/index.html
- Details in:
  - **macsim/doc/macsim.pdf**

Table 5. Source files and their purpose/content

File(s)	Purpose
frontend.cc/h, fetch_factory.cc/h, bp*.cc/h	Fetch stage
allocate*.cc/h, rob*.cc/h, map.cc/h	Decode and Allocate stages
schedule*.cc/h	Schedule stage
exec.cc/h	Execution stage
retire.cc/h	Retire stage
port.cc/h, cache.cc/h dram.cc/h, memory*.cc/h, memreq_info.cc/h, readonly_cache.cc/h, sw_managed_cache.cc/h	Memory system
pref*.cc/h	Prefetchers
trace_read.cc/h inst_info.h	Reading traces
core.cc/h	Class representing a core being simulated
process_manager.cc/h	Process Manager/thread scheduler
uop.cc/h	Uop structure and related enums
macsim.cc/h	Class containing pointers to the simulated cores, NoC, memory system, knobs and other objects
knob.cc/h	Classes for supporting knobs
statistics.cc/h	Classes for supporting knobs
factory_class.cc/h	Implementation of different factory classes
bug_detector.cc/h	Class useful for debugging forward progress errors happen
utils.cc/h	Utility classes and functions
debug_macros.h	Macros for debugging
assert_macros.h	Macros for assert statements with debug information
global*.h	Forward declarations and typedefs

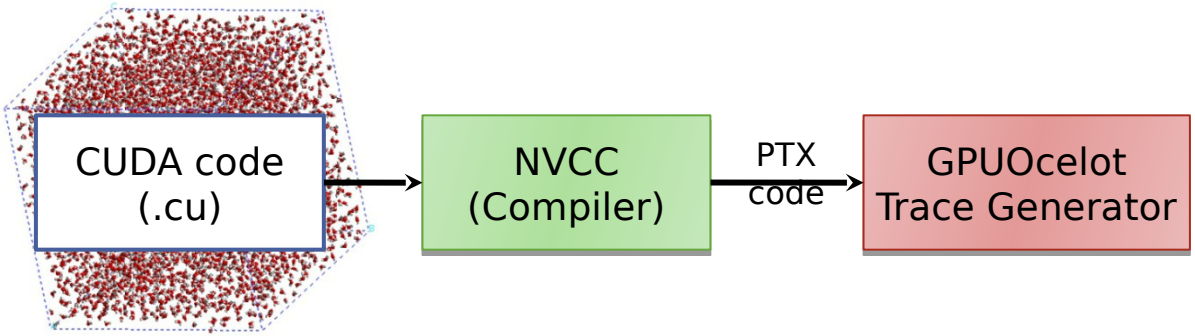


Trace Generation

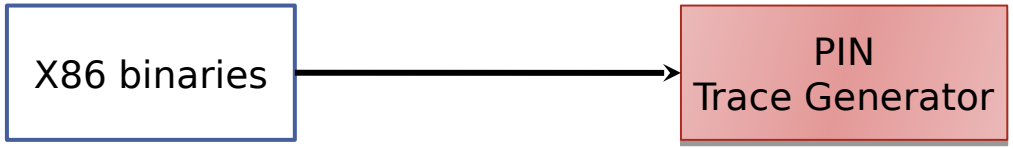
Georgia Tech comparch

Overview of Trace Generation

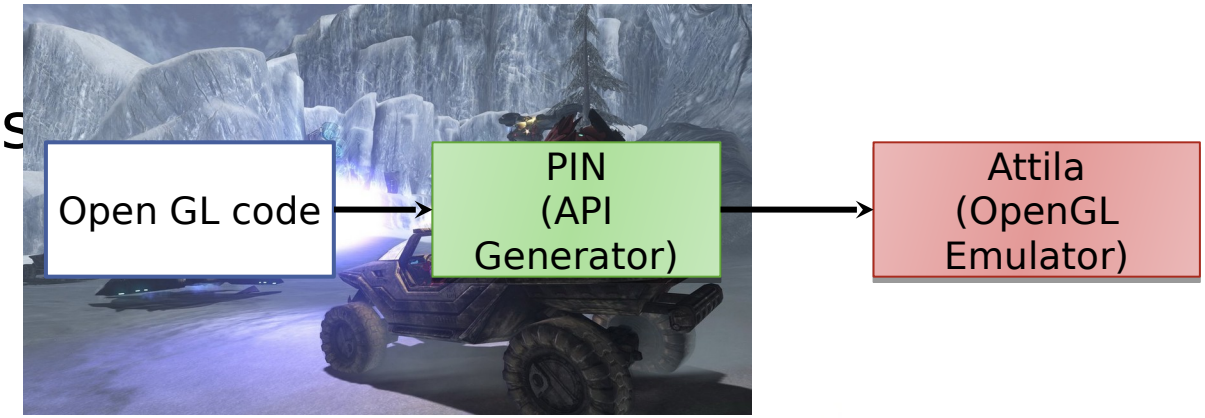
- GPGPU traces



- X86 traces



- Graphics traces (ongoing)



Trace Format

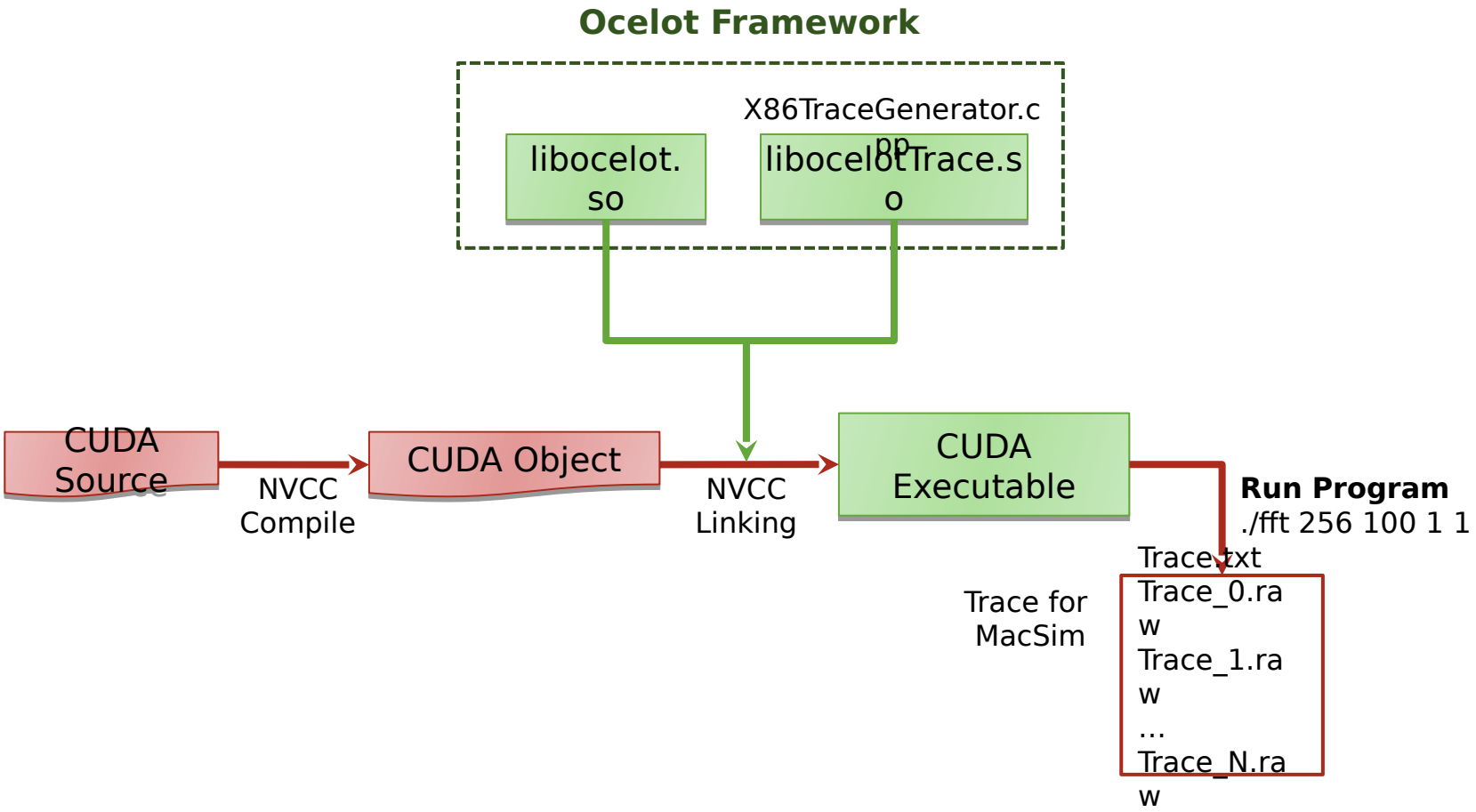
- Trace has following fields
  - PC, Opcode, Operands, Memory addresses, Memory request size, Active bits (to indicate divergent warp)
- Each thread (warp) has own file with its thread id and

```
Trace_0.raw          TID: 0      BID: 0
Trace_1.raw          TID: 1      BID: 0
Trace_65536.raw      TID: 65536  BID: 1
Trace_65537.raw      TID: 65537  BID: 1
```

- In MacSim, these trace streams are converted to internal RISC-style micro-ops (uop)

Overview of GPU Trace Generation

- MacSim uses **GPUOcelot** for GPU trace generation



Steps to Generate GPU Traces

- Please refer to
  - http://code.google.com/p/gpuocelot/wiki/TraceGeneration (Ocelot)
  - http://code.google.com/p/macsim/wiki/TraceGeneration (MacSim)
  - Documentation in MacSim repository

1. Checkout GPUOcelot from subversion repository

```
svn checkout http://gpuocelot.googlecode.com/svn/trunk gpuocelot
```

2. [illegible]

```
cd gpuocelot/ocelot; sudo ./build.py –install
cd gpuocelot/trace-generators; libtoolize; aclocal; autoconf; automake;
./configure; make; sudo make install
```

Steps to Generate GPU Traces – cnt'd

3. Link libocelot and libocelotTrace against CUDA applications

```
nvcc sourcefile.cu –locelot –ocelotTrace –lz –lboostxxx (boost libraries)
```

4. T

```
// configure.ocelot
x86Trace: true
```

e

5. R

```
./fft 256 100 1 1
```

Steps to Generate GPU Traces – cnt'd

Setting environment variables

- TRACE_PATH : directory to store traces (default: current dir)
- TRACE_NAME : prefix name for traces (default: Trace)
- KERNEL_INFO_PATH : file that contains kernel information
- COMPUTE_VERSION : compute capability

Example

```
In ~/.bashrc

# Create a trace directory in the /storage/traces
export TRACE_PATH="/storage/traces/"

# kernel_info has the kernel information
export KERNEL_INFO_PATH="kernel_info"

# Calculate occupancy based on compute capability 2.0
export COMPUTE_VERSION="2.0"
```

Steps to Generate GPU Traces – cnt'd

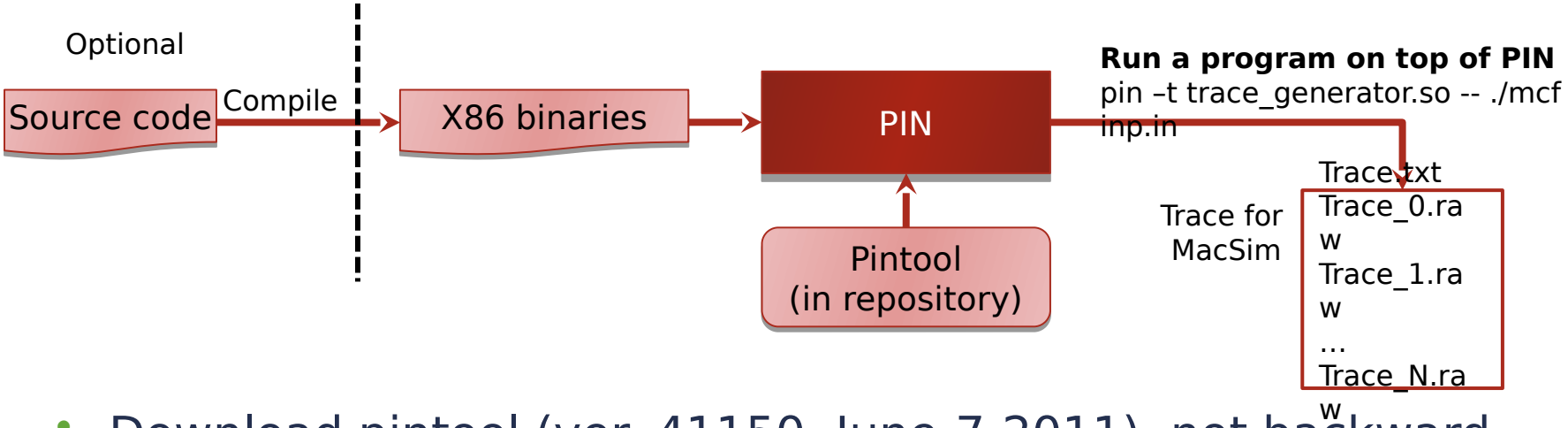
- The kernel information file (kernel_info) should contain the following information.
  - kernel name
  - register usage (per thread)
  - shared memory usage (per thread)
  - (e.g.) _Z9Memcpy_SWPfS_i 14 52

- Running CUDA application creates a directory with the kernel name where traces are generated as well as kernel_config.txt which is used for MacSim.

```
drwxr-xr-x   2 anonymous group 69632 Sep 00 22:03 _Z9Memcpy_SWPfS_i_0
-rw-r--r--   1 anonymous group    66 Sep 09 22:29 kernel_config.txt
```

Steps to Generate CPU Traces

MacSim uses **Pin** for CPU trace generation



- Download pintool (ver. 41150, June-7-2011), not backward compatible
- Bu m repository

```
cd macsim/tools/trace_generator; make
```

- Ru

```
pin –t trace_generator.so [options] – binary input
```

- **Pin homepage and tutorial: http://www.pintool.org**



Micro-ops (uops)

- Traces are translated to internal RISC-style micro-ops
  - All traces are shared same structure
  - Defined in trace_read.h and trace_read.cc
- GPU traces: almost 1-1 matching
- CPU traces: simple translation
  - information is coming from Pin's XED
  - XED provides Inst_category, register ids etc.
    - Does op have load/store?
    - Does op need a temp register?
    - Load+control flow, Load+store, load+no destination register
    - Does op have repeat?
    - Does op have a control flow instruction?



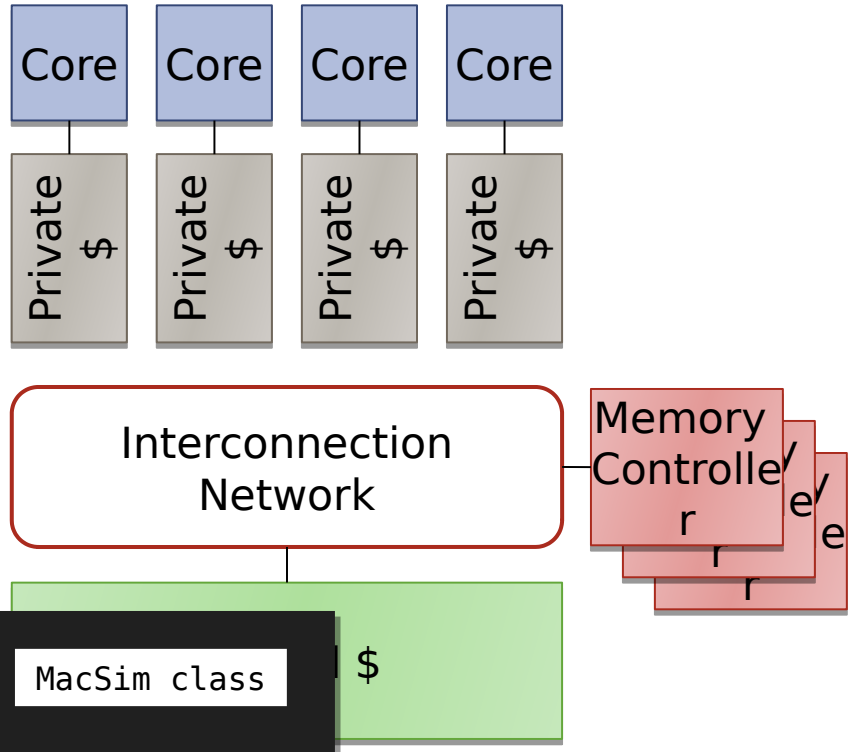
MacSim Memory System

Jaekyu Lee

Georgia Tech comparch

Overview of Memory System of MacSim

- Cache hierarchy
  - One or multiple levels
  - Highly flexible
- Interconnection Network
  - Default, IRIS
  - Ring, 2D mesh and torus
- Memory Controller
  - FCFS
  - FR-FCFS
- NoC and MCs are easily attachable/detachable with other modules

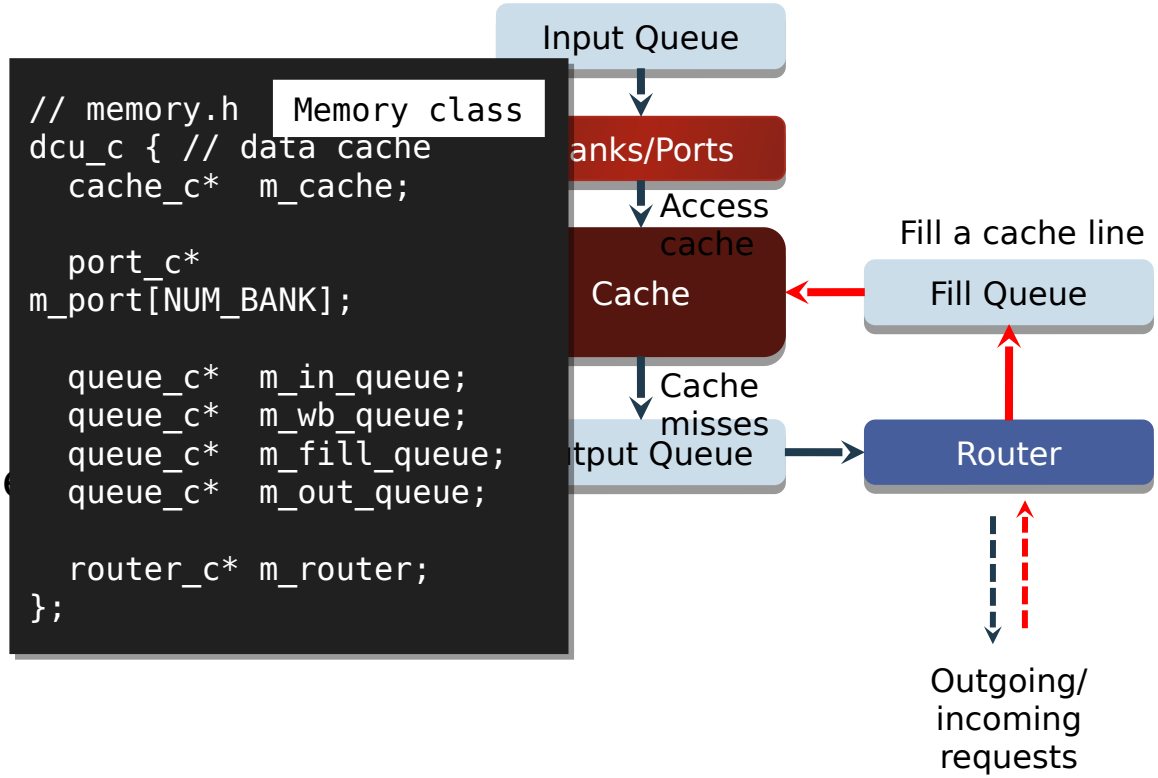



```
// macsim.h                    MacSim class
macsim_c {
  memory_c* m_memory;
  dram_controller_c**
m_dram_controller;
  router_wrapper_c* m_router;
};
```

Cache

- Each cache level consists
  - Cache
    - Banked
    - Multiple ports
  - Multiple queues
    - Input Queue
    - Output Queue
    - Write-back Queue
    - Fill Queue
  - Router

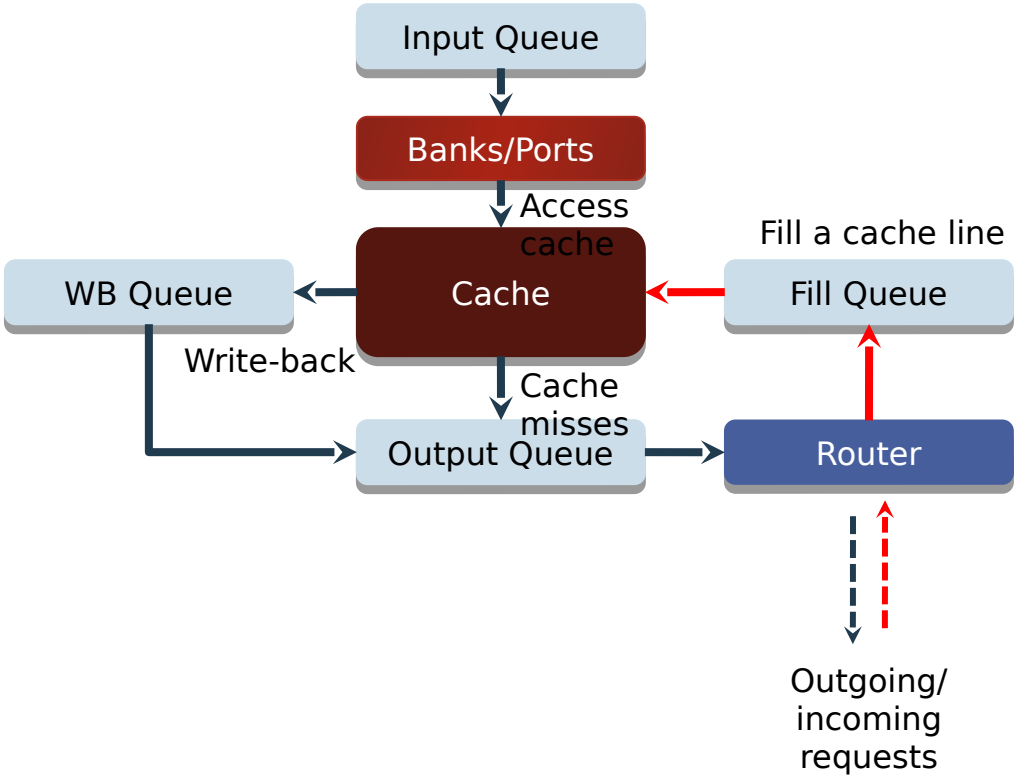
```
// memory.h   Memory class
dcu_c { // data cache
  cache_c*  m_cache;

  port_c*
m_port[NUM_BANK];

  queue_c*  m_in_queue;
  queue_c*  m_wb_queue;
  queue_c*  m_fill_queue;
  queue_c*  m_out_queue;

  router_c* m_router;
};
```



Cache – cnt'd

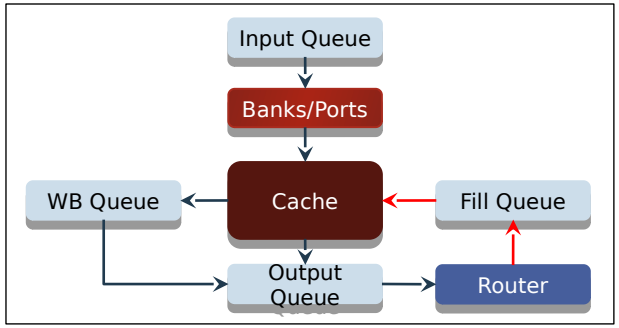
Cache operations and flows between queues

- Cache access
- Cache eviction
- Outgoing requests
- Cache line fill



How to Construct Cache Hierarchy

- To model various cache hierarchy models, we employ very flexible cache hierarchy



- We need to define
  - Cache has a router
    - If router is enabled, traffics between upper/lower caches are thru the router
  - Cache has a direct link between upper/lower cache
    - Otherwise, there must be a direct link between caches

How to Construct Cache Hierarchy – cnt'd

Cache constructor

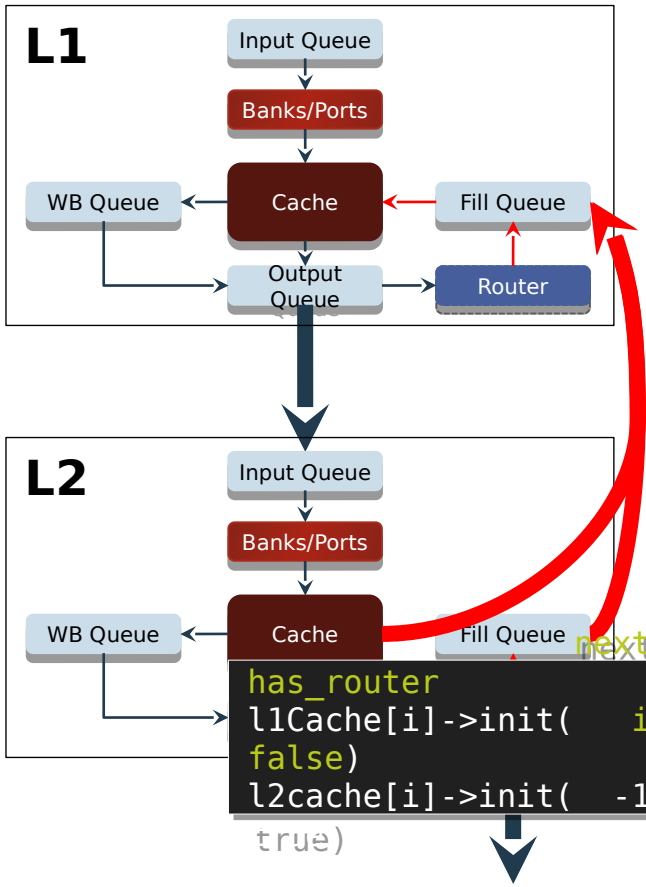
```
// cache constructor
dcu_c(
  int id,               // cache id
  Unit_Type type,       // core type
  int level,            // level
  memory_c* mem,        // pointer to the memory system
  int noc_id,           // router id
  dcu_c** next,         // pointer to upward cache
  dcu_c** prev,         // pointer to downward cache
  macsim_c* simBase);   // pointer to simulation base class
```

Cache initialization

```
// cache initialization function
void dcu_c::init(
  int next_id,         // downward cache id
  int prev_id,         // upward cache id
  bool coupled_up,     // has direct link to upward cache
  bool coupled_down,   // has direct link to downward cache
  bool disable,        // cache disabled?
  bool has_router);    // has router?
```

Cache Hierarchy Example - 1

- 2-level private cache with the direct link

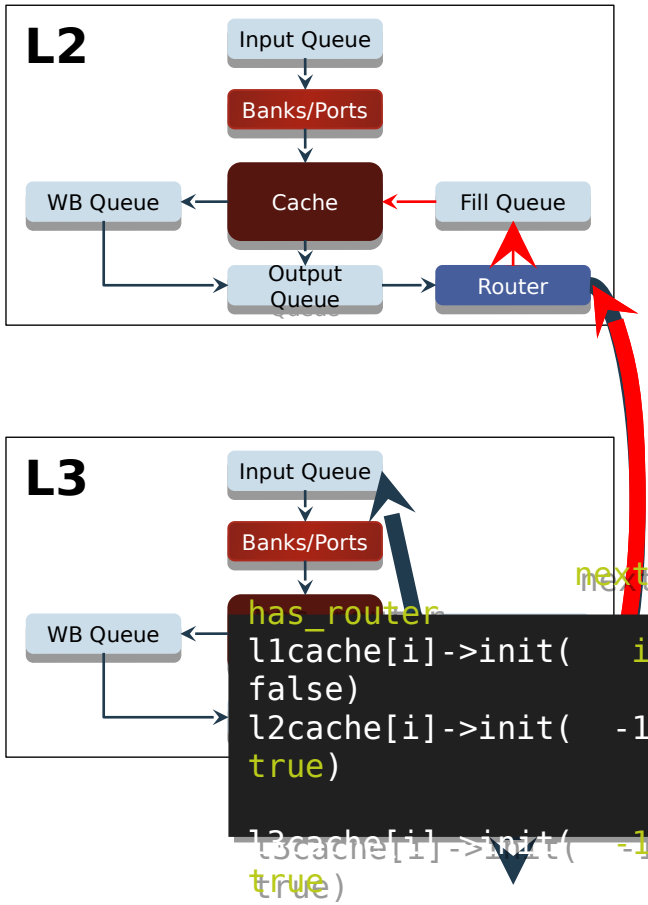


- L1 does not have router
- L1 miss
  - Inserted directly into L2's input queue
- L2 hit
  - Insert data directly into L1's fill queue
- L2 miss
  - Go thru L2's router

```
next_id, prev_id, coupled_up, coupled_down, disable
has_router
l1Cache[i]->init(   i,      -1,       false,       true,      false,
false)
l2cache[i]->init(   -1,      i,       true,        false,     false,
true)
```

Cache Hierarchy Example - 2

- Private L1, L2 caches with Shared L3 cache (No direct link)

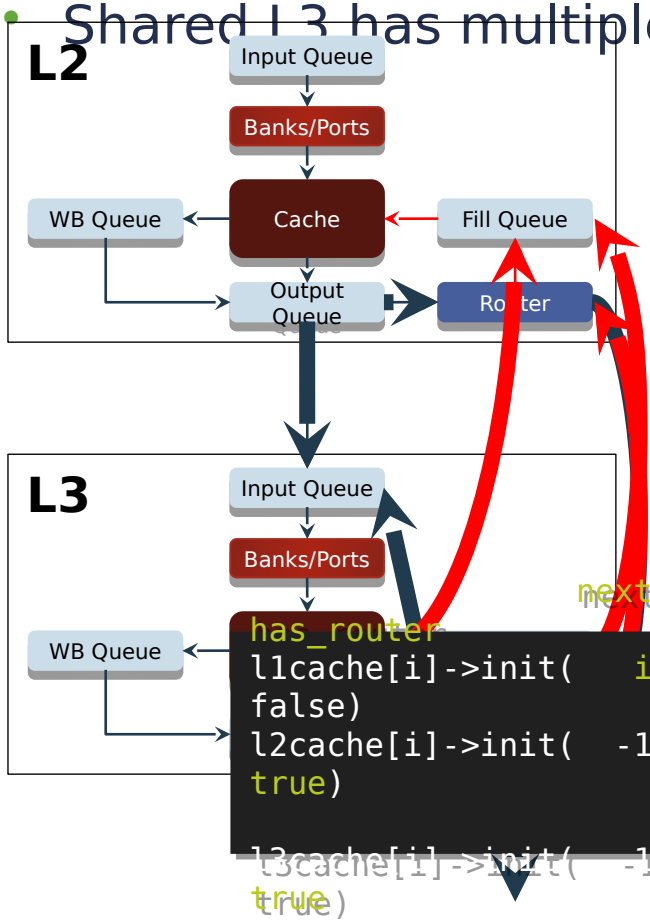


- L2 miss
  - Go thru L2's router
- L3 hit
  - Go thru L3's router to feed data to L2
- L3 miss
  - Go thru L3's router
  - L2 fill – Go thru L3's router to feed data to L2

```
next_id, prev_id, coupled_up, coupled_down, disable,
has_router
l1cache[i]->init(   i,      -1,      false,      true,      false,
false)
l2cache[i]->init(   -1,      i,      true,       false,     false,
true)

l3cache[i]->init(   -1,     -1,      false,      false,     false,
true)
```

Cache Hierarchy Example - 3

- Private L1, L2 caches with Shared L3 cache (With direct link)
  - Shared L3 has multiple tiles (bank)



- L2 miss
  - Destination: local tile – Use direct link
  - Destination: distant tile - Go thru L2's router
- L3 hit
  - Destination: local tile – use direct link
  - Destination: distance tile - Go thru L3's router to feed data to L2
- L3 miss

```
next_id, prev_id, coupled_up, coupled_down, disable,
l1cache[i]->init(   i,      -1,      false,      true,      false,
false)
l2cache[i]->init(   -1,      i,      true,      true,      false,
true)

l3cache[i]->init(   -1,      i,      true,      false,      false,
true)
```

Cache Hierarchy – cnt'd

- We provide sample cache hierarchies
  - Users need to specify single parameter to use sample classes
    - No cache - NVIDIA G80 GPU architecture
    - Private L1 / Shared L2 - NVIDIA Fermi GPU architecture, Intel Core architecture
    - Private L1, L2 / Shared+tiled L3 with direct link - Intel Sandy Bridge architecture
    - Private L1, L2 / Shared+tiled L3 without direct link - Generic 2D topology

```
Parameter: <string> KNOB_MEMORY_TYPE

class no_cache_c              // no cache architecture (NVIDIA G80)
class l3_coupled_network_c    // 3-level, direct link (Intel Sandy Bridge)
class l3_decoupled_network_c // 3-level, no direct link (General 2D
topology)
class l2_decoupled_network_c // 2-level, no direct link (NVIDIA Fermi)
```

Sample memory classes



Interconnection Network

- Default bi-direction ring network
  - Pipelined router architecture
  - Model virtual and physical channels with arbitrations
- IRIS interconnection simulator
  - CASL research group (Georgia Tech, led by Prof. Yalamanchili)
  - Model more detailed features
  - Have various topologies (Ring, 2D Mesh, Torus, …)

Memory Controller

- We model DRAM banks and channels
- DRAM bandwidth is modeled
- 3 timing parameters
  - Activate, Precharge, Column Access
- Scheduling policies
  - FCFS, FR-FCFS
  - Other policies can be easil
    implementable

```
// dram.h
class dram_controller_c {
  address_parsing
  dram_request_buffer,
  banks,
  channels,
  router,
  ...
};
```

DRAM controller pseudo code



Memory Access Handling

Request Merging

- Memory requests with the same address can be merged with other older requests in various locations
- Intra-core: MSHR
- Inter-core: DRAM request buffer

Coalesced vs. Uncoalesced

- If GPU SIMD threads generate memory requests contiguous address, multiple requests can be coalesced in a larger request
- We model coalescing in Trace Generator
- GPUOcelot provides memory address and access size for each thread of a warp (block)



Process Manager

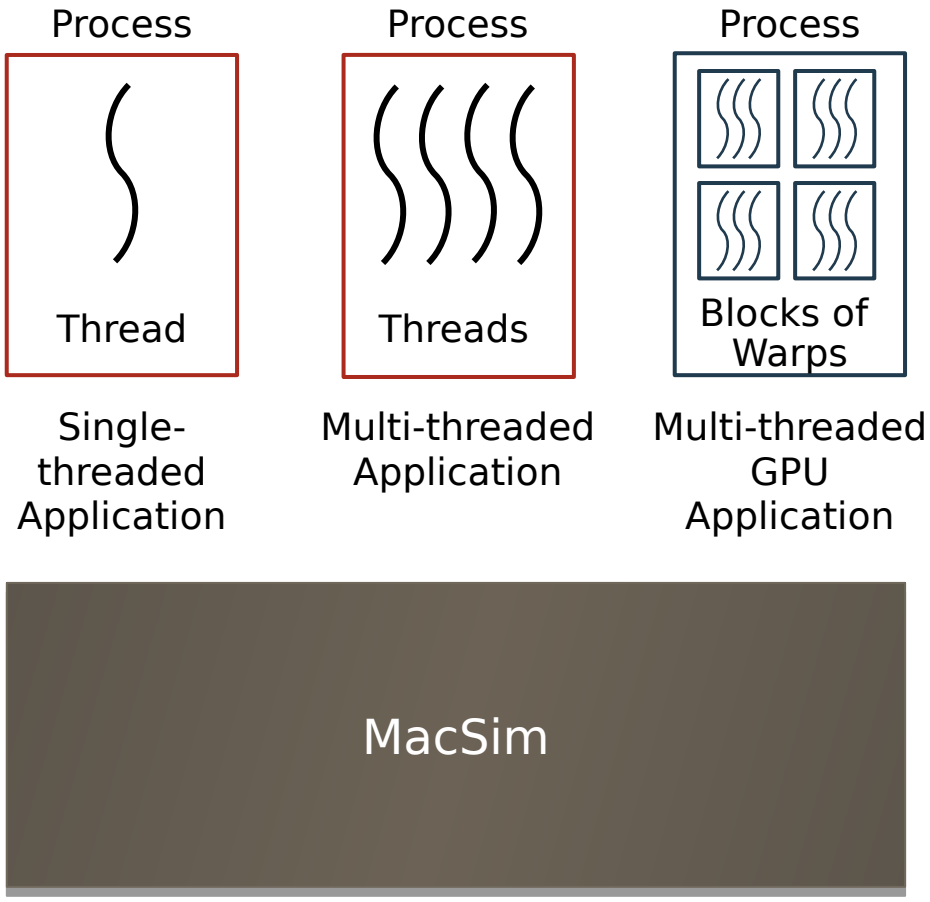
Georgia Tech comparch



Process Manager

- We use term the “Process” for an application in MacSim
- A Process consists of one or more threads (thread blocks)
- Process Manager is the component of MacSim that assigns work to cores
- It also handles creation/termination of process/threads (thread blocks)
  - Bookkeeping to track number of running threads, completed threads

Process Manager Capabilities



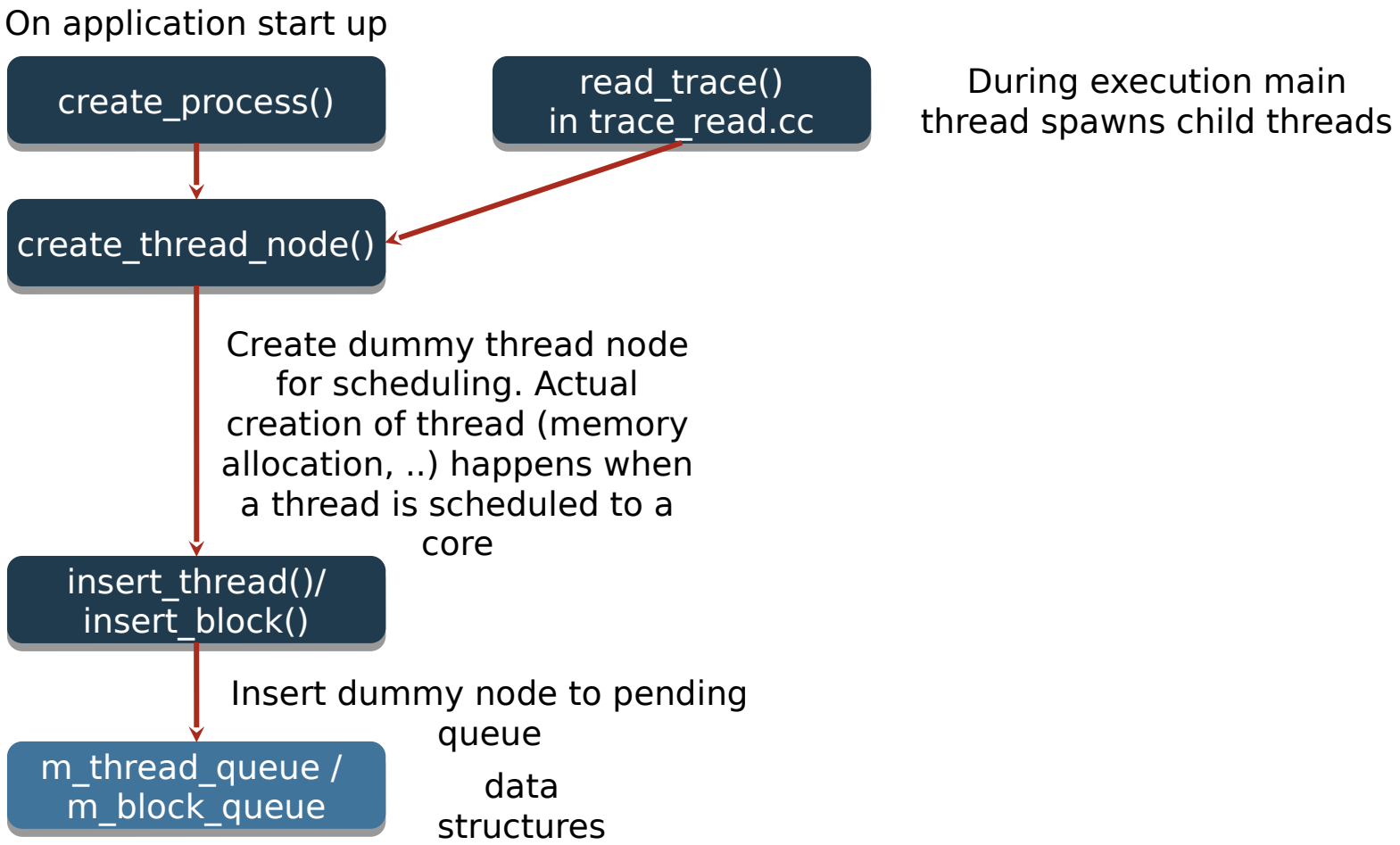
- Single process simulation
  - Single-threaded CPU
  - Multi-threaded CPU
  - GPGPU
- Multi-process simulation
  - Any combination of processes
- Configuring Multi-process simulations
  - Partition cores among applications
  - Specify the number of threads (thread blocks) per core
  - Specify repetition flag



Process/Thread Creation

- Allocate process structure, read trace config file and determine when threads have to be created
  - main thread created at start up
  - points of creation of child threads specified in terms of number of instructions executed by main thread
    - as the main thread executes check if a child thread should be created, if so, create the child thread and mark it ready for execution
  - for GPGPU applications all warps are created and marked ready at start up

Process/Thread Creation





Thread (Thread Block) Scheduling

CPU

- Greedy assignment of threads to cores
- Each core assigned threads up to user specified limit

GPU

- Scheduling at thread block granularity - all warps in a block assigned to same core
- Greedy assignment of blocks to cores (other policies can be added easily)
- Occupancy - How many blocks per core?
  - Calculated using Ocelot at trace generation and passed to Mascim via traces
  - Can be overridden by user

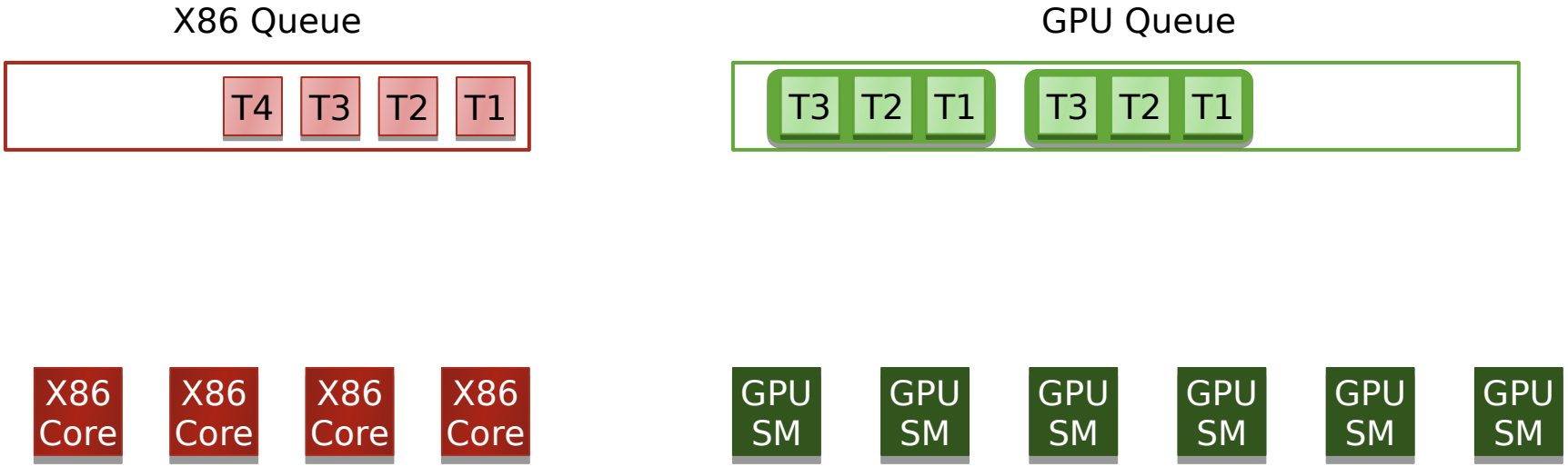
Once assigned threads/thread block remain on the same core until termination

Multi-application simulations (CPU+CPU or CPU+GPU),

- Each application is restricted to a statically defined set of cores
- Can repeat short running applications until a long running application ends

Thread Scheduling

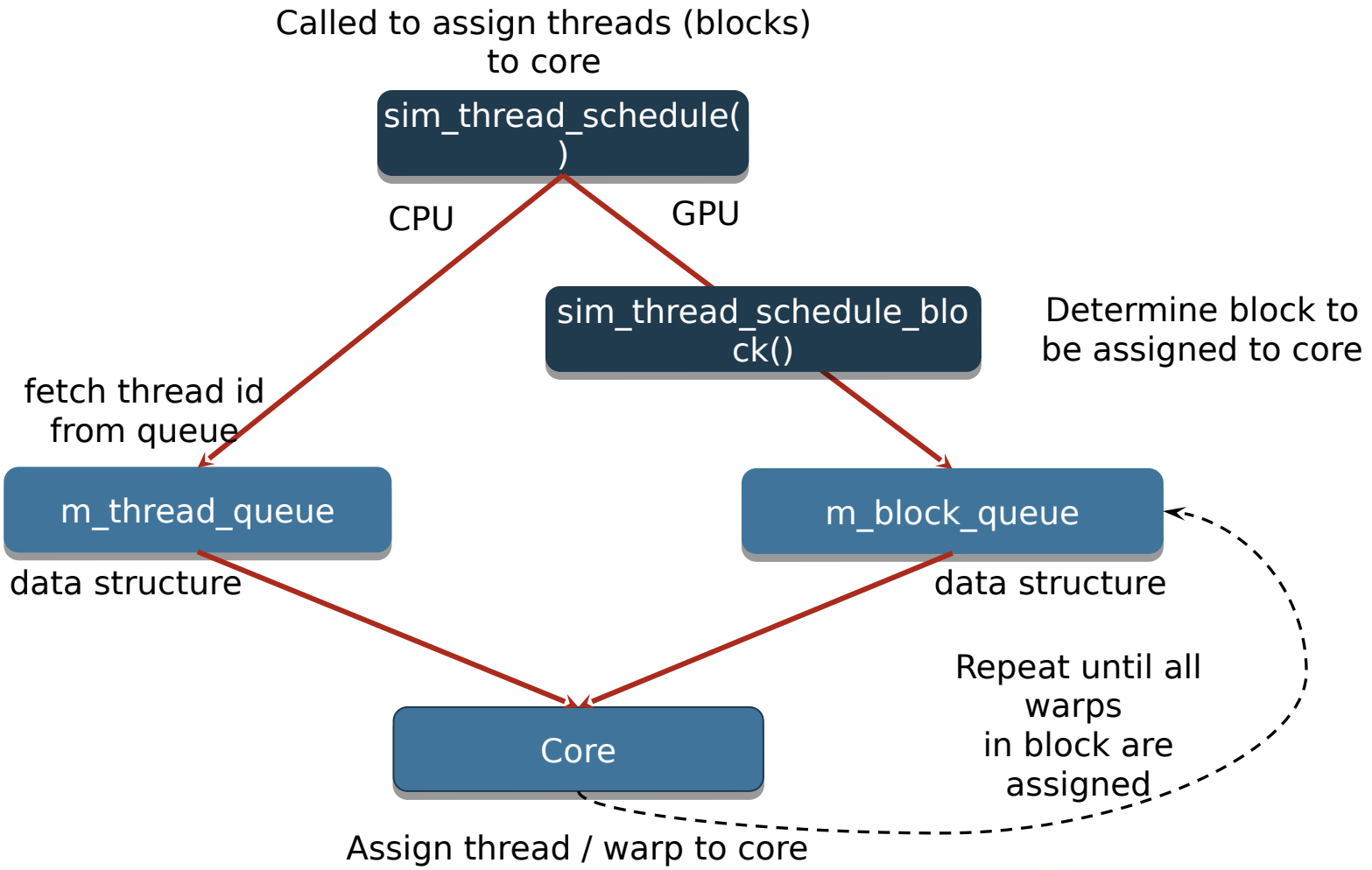
- Process Manager maintains queues of ready but unassigned threads/thread blocks



- Initially – greedy assignment to cores until specified limit
- Thereafter - assign a new thread to a core whenever a thread on the core terminates

- Initially – greedy assignment to cores until specified limit
- Thereafter - assign a new thread block to a core only when the core retires an entire thread block

Thread Scheduling





Process Manager

- Consists of two files
  - process_manager.h/cc
- process_manager.h
  - Declares Process Manager class
    - process_manager_c
  - Defines data structures for
    - Applications being simulated
      - process_s
    - Warps on GPU cores / threads on CPU cores
      - thread_s
    - Bookkeeping
      - block_schedule_info_s
      - thread_trace_info_node_s



Process Manager

- process_manager.cc
  - Defines member functions of process_manager_c class

Georgia Tech comparch



Process Manager

thread_s

- structure analogous to task structures used in OS kernels (task_struct in Linux)
- each CPU thread and GPU warp has an instance of thread_s
- contains fields for
  - thread id, block id, pointer to trace file, process to which thread/warp belongs, instructions read, trace buffer
  - In GPU cores, thread id represents warp id

Process Manager

thread_s

```
typedef struct thread_s {
  int                   m_thread_id; /**< current thread id */
  int                   m_block_id; /**< block id */
  gzFile                m_trace_file; /**< gzip trace file */
  bool                  m_file_opened; /**< trace file opened? */
  bool                    m_main_thread; /**< main thread (usually thread id 0)
*/
  uint64_t              m_inst_count; /**< total instruction counts */
  uint64_t              m_uop_count;  /**< total uop counts */
  bool                  m_trace_ended; /**< trace ended */
  process_s            *m_process; /**< point to the application belongs to */
  bool                  m_ptx; /**< GPU thread */
  char*                 m_buffer; /**< trace buffer */
  frontend_s*           m_fetch_data; /**< frontend fetch data */
  trace_info_s*         m_prev_trace_info; /**< prev instruction trace info */
  trace_info_s*         m_next_trace_info; /**< next instruction trace info */
  bool                  m_thread_init; /**< thread initialized */
  trace_uop_s*          m_trace_uop_array[MAX_PUP]; /**< trace uop array */
   trace_uop_s*           m_next_trace_uop_array[MAX_PUP]; /**< next trace uop
array */
} thread_s;
```



Process Manager

process_s

- one instance for each application being simulated
- common structure for both CPU and GPGPU applications
  - GPGPU applications do not include CPU-side traces, they only include traces from all kernels executed by the application
- has fields for process id, start info of threads, no. of threads (warps) in application, threads (warps) created, threads terminated (warps), list of valid cores for this application, list of kernels executed by application (GPGPU only)

Process Manager

- process_s

```
typedef struct process_s {
  process_s();
  ~process_s();

  unsigned int          m_process_id; /**< current process id */
  int                   m_orig_pid; /**< original process id - in case of repetition */
  int                   m_max_block; /**< max blocks per core for the application */
  thread_start_info_s *m_thread_start_info; /**< thread start information */
  thread_s**            m_thread_trace_info; /**< thread trace information */
  unsigned int          m_no_of_threads; /**< number of total threads */
  unsigned int          m_no_of_threads_created; /**< number of threads created */
  unsigned int          m_no_of_threads_terminated; /**< number of terminated threads */
  map<int, bool>        m_core_list; /**< list of cores that this process is executed */
  queue<int>           *m_core_pool; /**< core pool pointer */
  bool                  m_ptx; /**< GPU application */
  int                   m_repeat; /**< application has been re-executed */
  vector<string>        m_applications; /**< list of sub-applications */
   vector<int>           m_kernel_block_start_count; /**< block id start count for sub-
appl. */
  string                m_current_file_name_base; /**< current sub-appl.'s filename base
*/
  string                m_kernel_config_name; /**< kernel config file name */
   int                          m_current_vector_index; /**< current index to the sub-
application */
  map<int, bool>        m_block_list; /**< list of block currently running */
  uns64                 m_inst_count_tot; /**< total instruction counts */
  int                   m_block_count; /**< total block counts */
} process_s;
```



Process Manager

block_schedule_info_s

- bookkeeping structure for thread blocks in a GPGPU application, one instance for each thread block
- contains fields for no. of warps in block, no. of terminated warps, core to which block is assigned

```
typedef struct block_schedule_info_s {
  bool    m_start_to_fetch;     /**< start fetching */
   int      m_dispatched_core_id; /**< core id in which this block is launched
*/
  bool    m_retired;            /**< retired */
  int     m_dispatched_thread_num; /**< number of dispatched threads */
  int     m_retired_thread_num; /**< number of retired threads */
  int     m_total_thread_num;   /**< number of total threads */
  int     m_dispatch_done;      /**< dispatch done */
  bool    m_trace_exist;        /**< trace exist */
  Counter m_sched_cycle;        /**< scheduled cycle */
  Counter m_retire_cycle;       /**< retired cycle */
} block_schedule_info_s;
```



Process Manager

thread_trace_info_node_s

- wrapper structure around thread_s used by process_manager_c to track threads (warps) yet to be assigned to cores

```
typedef struct thread_trace_info_node_s {
  thread_s*  m_trace_info_ptr;  /**< trace information pointer */
  process_s* m_process;         /**< pointer to the process */
  int        m_tid;             /**< thread id */
  bool       m_main;            /**< main thread */
  bool       m_ptx;             /**< GPU simulation */
  int        m_block_id;        /**< block id */
} thread_trace_info_node_s;
```



Process Manager

process_manager_c

- creates processes, threads (warps)
- does assignment of threads / thread blocks to cores
- invoked again on termination of threads (warps)
- contains list of threads not assigned to any core (CPU only)
- contains list of threads blocks not assigned to any core (GPGPU only)
- handles repetition of applications (triggered by retire) when multiple applications are being run



Process Manager

Process/Thread creation and termination

```
// allocates process_s node, reads config file and determines
no.
// of threads/warps and thread blocks in application / each
kernel
// of application, start info of threads
create_process()

// called for each thread/warp when it becomes ready for
launch,
// allocates a thread_trace_info_node_s node and inserts into
// m_thread_queue or m_block_queue. All warps are ready for
launch
// at the start of a kernel
create_thread_node()

// allocates and initializes a thread_s node, opens trace
file
// for thread/warp
create_thread()

// cleans up some data structures and saves stats (if this
the
// first run of the application)
terminate_process()

// clears data structures, retires thread block if all warps
in
// block have completed, calls sim_thread_schedule()
terminate_thread()
```

Process Manager

- Thread/block scheduling

```
// assigns a thread to a core or a thread block to a core if
the
// number of threads/blocks on the core are fewer than the
maximum
// allowed
void process_manager_c::sim_thread_schedule(void);

// called by sim_thread_schedule() to determine the next
block
// that will be assigned to a core
int           process_manager_c::sim_thread_schedule_block(int
core_id);

// inserts thread / thread_block into list of threads /
blocks yet
// to be launched
void
process_manager_c::insert_thread(thread_trace_info_node_s*)
void
process_manager_c::insert_block(thread_trace_info_node_s*)
```



Classes/Data Structures

Nagesh Lakshminarayana

Georgia Tech comparch

Data Structures

- uop_c
  - main data structure for uops
    - an instance is allocated for each uop
  - contains thread (warp) id, core id, block id (GPU only), timestamps of when uop passed through different stages, opcode type, source and destination operands and all other information required for execution
    - contains list of child uops in case of uncoalesced memory accesses

- New uops are allocated from a uop pool and completed uops are returned to the pool



Data Structures

core_c

- class for cores in simulation
- contains pointers to objects for pipeline stages and components
- during simulation run_a_cycle() of core_c is called every cycle
  - core_c::run_a_cycle() calls run_a_cycle() function of the pipeline stages

Data Structures

map_c

- Class for tracking data and memory dependences between uops of a thread
- once a dependence between uops is identified, pointer to source uop is added to the list of uops on which the dependent will wait
  - in the schedule stage a check is made to see if all sources of a uop have completed

Data Structures

mem_req_s

- data structure for memory requests
  - an instance of mem_req_s is allocated for every memory request
- contains id information, details of primary memory request and piggybacking requests, pointer to done function
  - done function is a callback function called when the memory request is completed (actually when response reaches L2)
    - user can add code here for collecting statistics



Data Structures

drb_entry_s

- data structure for requests in DRAM
  - contains timestamp, row id, bank id and pointer to memory request
- requests to a bank are held in a list, DRAM scheduler sorts this list according to policy being implemented and returns the head of the list
  - can extend the structure to track any information that your DRAM policy requires



Utility Classes

Nagesh Lakshminarayana

Georgia Tech comparch



Utility Classes

pool_c

- class used to implement memory pools of objects/strctures of different types
  - acquire_entry() and release_entry() for getting/releasing an from/to the pool
  - pool expands automatically when it is out of space

pqueue_c

- class for modeling latencies of pipeline stages
  - entries pushed into the queue are ready to be removed from queue after the specified latency
  - priority used among entries pushed into queue in the same cycle



Utility Classes

Factory classes

- Fetch policies, DRAM policies, memory hierarchies and so on are implemented by classes whose objects are instantiated via factory mechanisms
  - One factory class for fetch policies, one for DRAM policies, …
- For supporting new policies
  - Define a class for the policy
  - Register class with factory under the name of the policy
  - Specify new policy as the policy to be used during simulation (via simulation parameters)



Knob Variables

Nagesh Lakshminarayana

Georgia Tech comparch



Knobs

- Used to specify architecture and simulation parameter values
- Available knobs can be found in .param.def files in def/ directory under main source directory
- To set a knob
  - provide a value on the command line (highest priority),
  - provide a value in the parameter file, or
  - provide a default value in the definition (lowest priority)
- All knobs and their values for a simulation are written to *params.out* file when simulation starts



Adding New Knobs

- Add a definition in a relevant .param.def file in def/ directory (create a new file if necessary)
  - Knob definitions converted into c++ source and included in build process automatically

- Format of definition
  - param<{name_in_code}, {name}, {knob_type}, {default_value}>
    - name_in_code – name for accessing knob in the code (in upper case)
    - name – name used in command line or parameter file (in lower case)
    - knob_type – bool, different int types, float, string
    - default value must be provided

- Example definition
  - param<L2_ASSOC, l2_assoc, int, 8>



Setting and Using Knobs

Example definition

- param<L2_ASSOC, l2_assoc, int, 8>

Setting value on command line

- ./macsim --l2_assoc=16

Setting value in parameter file (params.in)

- l2_assoc 16

Accessing in code

- use KNOB_L2_ASSOC (prefix knob name with KNOB_)
  - *m_simBase->m_knobs->KNOB_L2_ASSOC, or
  - m_simBase->m_knobs->KNOB_L2_ASSOC->getValue()



Statistics

Nagesh Lakshminarayana

Georgia Tech comparch



Stats

- Global or per core stats
- Three Stat types supported
  - COUNT
    - #occurrences of an event
  - RATIO
    - Ratio of #occurrences of event A to the #occurrences of event B
  - DIST
    - Proportion of each event in a group of events
- Stat definitions can be found in .stat.def files in def/ directory under main source directory



Stats

- At the end of simulation .stat.out files containing stat values are generated
  - single application simulation - .stat.out files
  - Multi-application simulation - .stat.out.<appl_id> files
    - stat files generated when the first run of application completes
- Script to read stat values from the stat files will be provided
  - provide name of stat and value of stat will be printed
  - script can also evaluate simple expressions containing stat names

COUNT Stat

Format of definition

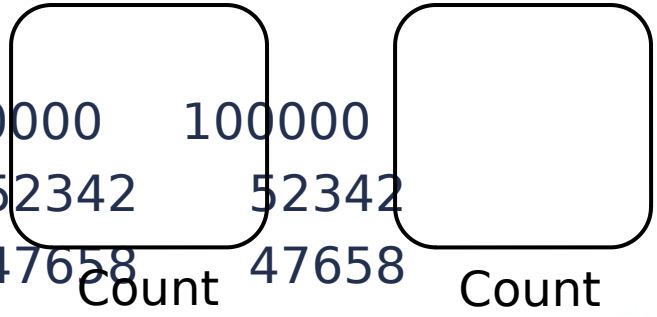
- DEF_STAT(*STAT_NAME*, COUNT, NO_RATIO [, PER_CORE])
  - STAT_NAME – name for accessing stat in code (in upper case)
  - NO_RATIO – not a ratio stat
  - PER_CORE – optional, use if you want a per core stat

Example definitions

- DEF_STAT(INST_COUNT_TOT, COUNT, NO_RATIO)
- DEF_STAT(INST_COUNT, COUNT, NO_RATIO, PER_CORE)

Example output

- INST_COUNT_TOT 100000 100000
  INST_COUNT_CORE_0 52342 52342
  INST_COUNT_CORE_1 47658 47658



RATIO Stat

Format of definition

- DEF_STAT(*STAT_NAME*, RATIO, *BASE_STAT_NAME* [, PER_CORE])
  - STAT_NAME – name for accessing stat in code (in upper case)
  - BASE_STAT_NAME – stat whose value is the denominator in the ratio
  - PER_CORE – optional, use if you want a per core stat
  - Requires the definition of a stat of type COUNT

Example definition

- DEF_STAT(DCACHE_ACCESSES, COUNT, NO_RATIO)
  DEF_STAT(DCACHE_HITS, RATIO, DCACHE_ACCESSES)

Example output

- DCACHE_ACCESSES 67258 67258
  DCACHE_HITS 8434 0.123

Count

Count

Ratio = (8434 / 67258) = (#DCACHE_HITS/#DCACHE_ACCESSES)

DIST Stat

Format of definition

- DEF_STAT(*STAT_NAME_START*, DIST, NO_RATIO [, PER_CORE])
  DEF_STAT(*STAT_NAME*, COUNT, NO_RATIO [, PER_CORE])*
  DEF_STAT(*STAT_NAME_END*, DIST, NO_RATIO [, PER_CORE])
  - DIST stat should contain at least two stats

Example definition

- DEF_STAT(ICACHE_HITS, DIST, NO_RATIO)
  DEF_STAT(ICACHE_MISSES, DIST, NO_RATIO)

Sample output

- ICACHE_HITS 292331 0.8937
  ICACHE_MISSES 34767 0.1063

Count

$\Sigma(prop_i) = 1 = 100\%$

Proportion = (292331 /(292331 + 34767)) = 89.37%

Proportion = (34767 /(292331 + 34767)) = 10.63%



Updating Stats

Macros for updating Stats

- Global stats
  - STAT_EVENT(STAT_NAME) – increment STAT_NAME by 1
  - STAT_EVENT_N(STAT_NAME, n) – increment STAT_NAME by n
  - STAT_EVENT_M(STAT_NAME) – decrement STAT_NAME by 1

- Per-core stats
  - STAT_CORE_EVENT(CORE_ID, STAT_NAME) – increment STAT_NAME for core CORE_ID by 1
  - STAT_CORE_EVENT_N(CORE_ID, STAT_NAME, n) – increment STAT_NAME for core CORE_ID by n
  - STAT_CORE_EVENT_M(CORE_ID, STAT_NAME,) – decrement STAT_NAME for core CORE_ID by 1



Debugging

Nagesh Lakshminarayana

Georgia Tech comparch



Debug Statements

- Debug statements can be printed only with the debug version of the MacSim binary
  - Run “make dbg” to build the debug version
- To output debug statements for a stage/component
  - Set knob debug_cycle_start to the cycle value from when debug statements should be printed
    - Cycle value should be greater than zero
  - Set the debug enable knob for the stage/component to 1
    - Knobs available for pipeline stages, memory system and so on
    - Check debug.param.def for list of available knobs
- Knob debug_cycle_stop can be set to turn off debug statements after some point
  - Set debug_cycle_stop to zero (default) to print till end of simulation

Debug Statements

- params.in file with debugging enabled for front-end stage
  - ...

    debug_cycle_start 1

    debug_cycle_stop 0

    debug_front_stage 1

    ...



Adding New Debug Statements

- Debug statements can be added using the DEBUG macro
- DEBUG macro takes parameters similar to the printf() function
- Example debug statement
  - DEBUG("m_core_id:%d thread_id:%d uop_num:%lld operands are not ready \n",
    m_core_id, cur_uop->m_thread_id, cur_uop->m_uop_num);

Georgia Tech comparch

Sample Debug output

```
../src/frontend.cc:836: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 m_running_thread_num:1 m_fetching_thread_num:1 m_uniqu
e_scheduled_thread_num:1
../src/frontend.cc:914: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 try_again:1 tid:0
../src/frontend.cc:230: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 frontend fetch thread is:0
../src/frontend.cc:633: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 fetch_addr:46c856 new fetch_addr:46c856 m_icache hit
../src/frontend.cc:465: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  cycle_count:1136 m_core_id:0 tid:0 uop_num:60  inst_num:50 uop.va:
b54658 iaq:0 mem_type:2 dest:0 num_dests:0
../src/frontend.cc:550: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 tid:0 MT_scheduler[0]->0x46c85d
../src/frontend.cc:721: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 tid:0 inst_num:50 uop_num:60 opcode:11 isitEOM:1 sent
to qfe
../src/frontend.cc:465: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  cycle_count:1136 m_core_id:0 tid:0 uop_num:61  inst_num:51 uop.va:
0 iaq:0 mem_type:0 dest:19 num_dests:1
../src/frontend.cc:550: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 tid:0 MT_scheduler[0]->0x46c867
../src/frontend.cc:721: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1136):  m_core_id:0 tid:0 inst_num:51 uop_num:61 opcode:11 isitEOM:1 sent
to qfe
../src/frontend.cc:836: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 m_running_thread_num:1 m_fetching_thread_num:1 m_uniqu
e_scheduled_thread_num:1
../src/frontend.cc:914: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 try_again:1 tid:0
../src/frontend.cc:230: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 frontend fetch thread is:0
../src/frontend.cc:633: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 fetch_addr:46c867 new fetch_addr:46c867 m_icache hit
../src/frontend.cc:465: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  cycle_count:1137 m_core_id:0 tid:0 uop_num:62  inst_num:52 uop.va:
64203e8 iaq:0 mem_type:2 dest:0 num_dests:0
../src/frontend.cc:550: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 tid:0 MT_scheduler[0]->0x46c86b
../src/frontend.cc:721: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 tid:0 inst_num:52 uop_num:62 opcode:11 isitEOM:1 sent
to qfe
../src/frontend.cc:465: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  cycle_count:1137 m_core_id:0 tid:0 uop_num:63  inst_num:53 uop.va:
64203f0 iaq:0 mem_type:2 dest:0 num_dests:0
../src/frontend.cc:550: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 tid:0 MT_scheduler[0]->0x46c86f
../src/frontend.cc:721: *m_simBase->m_knobs->KNOB_DEBUG_FRONT_STAGE (I=48  C=1137):  m_core_id:0 tid:0 inst_num:53 uop_num:63 opcode:11 isitEOM:1 sent
to qfe
```



Forward Progress Error

- A Forward Progress Error is triggered when an active core has not retired any instructions for the number of cycles specified by the knob forward_progress_limit
  - A Forward Progress Error causes simulation to be aborted

- A Forward Progress Error could be triggered when
  - A uop cannot be retired because the memory request it generated was lost in the memory system
  - A uop cannot be scheduled because one of its sources operands is not marked ready yet, but the uop producing the source has already retired (or is lost)



Bug Detector

- The knob bug_detector_enable can be turned on to generate additional debug information that would be helpful in resolving forward progress errors
  - Works with both debug and optimized versions of binary
  - Generates
    - bug_detect_uop.out
      - Dump of all uops in the pipeline of the failing core
    - bug_detect_mem.out
      - Dump of all memory requests issued by the failing core
    - bug_detect_dram.out
      - Dump of all requests in DRAM



Questions?