SMT

2017年12月26日 18:59

Physical thread => 对应core.h phythread[threadnum] var

Structure

表示硬件上的物理thread实体

<table>
<tr><td>Cnt</td><td></td></tr>
<tr><td>Fe_active</td><td></td></tr>
<tr><td>Al_active</td><td></td></tr>
<tr><td>Available</td><td></td></tr>
<tr><td>Thread_using</td><td></td></tr>
<tr><td>Cid</td><td></td></tr>
<tr><td>Context[MAX_CONTEXT_NUM]</td><td>表示当发生mispredict的情况下，function model可以执行的最大的错误分支路径
包括 rat_table，partial_flag
Context_s
<table><tr><td>Tid</td><td></td></tr><tr><td>Phytid</td><td></td></tr><tr><td>Prev_cid</td><td></td></tr><tr><td>Next_cid</td><td></td></tr><tr><td>Depth</td><td></td></tr><tr><td>Prev_rob</td><td></td></tr><tr><td>Prev_store_tail</td><td></td></tr><tr><td>Al_active</td><td></td></tr><tr><td>Rat</td><td>Mapping</td></tr><tr><td>Partial_flags</td><td></td></tr><tr><td>Simrunner</td><td></td></tr><tr><td>Stew</td><td></td></tr><tr><td>Bigstew</td><td></td></tr><tr><td>Ras_stack</td><td></td></tr><tr><td>Ras_tos</td><td></td></tr><tr><td>Call_depth</td><td></td></tr><tr><td>Global_history</td><td></td></tr><tr><td>Target</td><td>VA</td></tr><tr><td>Last_br</td><td>CpuframeworkUop</td></tr><tr><td>Reached_rat</td><td></td></tr></table></td></tr>
<tr><td>Clonenum</td><td></td></tr>
<tr><td>Deletenum</td><td></td></tr>
<tr><td>Context_active_list</td><td></td></tr>
<tr><td>Context_free_list</td><td></td></tr>
<tr><td>Fe_mode</td><td></td></tr>
<tr><td>Fe_prev_mode</td><td></td></tr>
<tr><td>Fe_current_instr_disasm</td><td></td></tr>
<tr><td>Fe_current_instr_bytes</td><td></td></tr>
<tr><td>Fe_fetch_plr</td><td></td></tr>
<tr><td>Plr_id</td><td></td></tr>
<tr><td>Fe_fetch_stall</td><td></td></tr>
<tr><td>Fe_icache_itlb_stall</td><td></td></tr>
<tr><td>Fe_fetch_stall_reason</td><td></td></tr>
<tr><td>Fe_stall_btb_notstalled</td><td></td></tr>
<tr><td>Fe_uq_stall</td><td></td></tr>
<tr><td>Is_64bit_mode</td><td></td></tr>
<tr><td>Fe_transition_point</td><td></td></tr>
<tr><td>Fe_restart</td><td></td></tr>
<tr><td>Fe_sild_mode</td><td></td></tr>
<tr><td>Fe_seen_prefix_bubble</td><td></td></tr>
<tr><td>Uq_is_empty</td><td></td></tr>
<tr><td>Fe_current_instr</td><td>CpuframeworkInstr_s</td></tr>
<tr><td>Fe_prev_first_uop</td><td>CpuframeworkUop_s</td></tr>
<tr><td>Cycle_last_clear</td><td></td></tr>
<tr><td>Fe_sb_occupied</td><td>Sb=stream-buffer</td></tr>
<tr><td>Fe_sb_excellent_line</td><td></td></tr>
<tr><td>Fe_stream_sent</td><td></td></tr>
<tr><td>Fe_mtf_lip</td><td></td></tr>
</table>

Fe_il1_line	
Fe_seen_ms	
Fetch_ul2_checked	
Fe_sild_addr	
Fe_in_irq	
Fe_excellent_address	Va
Fe_excellent_address_paddr	Pa
Fe_last_btflush	Va
Fe_btflush_queue	
Inst_had_il1_miss	
Fe_previous_pos_in_line	
Fe_previous_line_number	
Fe_trace_plr	
Ms_ignore_fusing	
Itlb_miss	
Itlb_write	
Miss_in_itlb	
Tid	
Uops_fetched	
Uops_scheduled	
Uops_retired	
Al_stall	
Al_stall_bubble	
Mwaiting	
Block_younger_loads	
Look_for_store_unlock	
Lock_uop_num	
Lock_paddr	VA?
Lock_vaddr	Va
Num_locks	
Last_uop_retire_time	
Last_inst_retire_time	
Last_retired_uop_alloc_time	
Last_retired_uop_fetch_time	
Earliest_next_retire_time	
Last_retired_uop_action	Beuflush_action_t
Stalled_fro_rob_entries	

Physical thread selection policy

POLICY_INDEPENDENT
POLICY_PHYTHREAD_INTERLEAVE
POLICY_PHYTHREAD_ROUNDROBIN
POLICY_ICOUNT
POLICY_OLDEST_FIRST
POLICY_LEAST_RECENTLY_SERVICED
POLICY_REPLICATE
POLICY_TIME_INTERLEAVE
POLICY_THREAD_UNAWARE

Logical thread => 对应core.h thread[threadnum] var

表示OS看到的logic CPU的实体

State	THREAD_STATE THREAD_ACTIVE THREAD_READY THREAD_STALLED THREAD_ACTIVE_IDLE
Active	
Active_stall	
Phytid	
Last_phytid	
Num_stall_events	
Imiss_stall	
Cycle_last_end_drain	
Cycle_last_activate	
Cycle_last_inactivate	
Uop_fetch_last_inactivate	
Uop_retire_last_activate	
Cycles_last_act	
Retry_cycle	
Al_prevent_switch	

Miss_vaddr	VA
Miss_paddr	PA

Core的physical thread的轮询模板

ThreadState

```
struct ThreadState {
  int phytid;                    // 当前T正在处理的physical thread
  int save_g_phytid;             // 当前T开始轮询前的初始physical thread，这个变量由global_thread_priority或是g_phytid[all_core][phytid_idx]决定
  int count;                     // 当前T正在处理的physical thread处理了多少请求
  int phytids_used_count;        // 当前T已经有多少个physical thread已经处理过了
  int phytids_used[MAX_THREADS]; // 当前T哪些physical thread处理了，哪些没有处理
  float min_used;                // 资源使用情况，只记录占用量最小的，(float)used_resource/limit_resource
};
```

g_phytid[all_core][phytid_idx]是个函数内的static变量，用于保存当前T执行完后，下一个T用于获取初始physical thread的信息，更新规则为：如果当前T有处理请求，则把最后服务的physical thread记录到phytid_idx(仅仅是轮询时用的索引变量，与physical thread无关)的索引中

目前physical thread轮询支持的Policy

```
enum {
  POLICY_INDEPENDENT;
  // 遍历所有的physical thread，只要满足处理条件全部处理
  POLICY_PHYTHREAD_INTERLEAVE;
  // 找出满足条件的physical thread，处理一次，如果没有处理，则查找下个满足条件的physical thread，直到有一个可以处理或是遍历所有physical thread
  POLICY_PHYTHREAD_ROUNDROBIN;
  // 当前T遍历所有的满足条件的physical thread
  POLICY_ICOUNT;
  // 首先找出使用资源最少的phyiscal thread进行处理；如果ThreadState.count已经处理过了，则当前T结束，否则轮询下个资源使用最少的physical thread
  POLICY_OLDEST_FIRST;
  // 同PHYTHREAD_ROUNDROBIN
  POLICY_LEAST_RECENT_SERVICED;
  // 同PHYTHREAD_ROUNDROBIN
  POLICY_REPLICATED;
  // 同PHYTHREAD_ROUNDROBIN
  POLICY_TIME_INTERLEAVE;
  // 只处理当前初始选择的physical thread，无论处理条件是否满足
  POLICY_THREAD_UNWARE;
  // 同PHYTHREAD_ROUNDROBIN
};
```