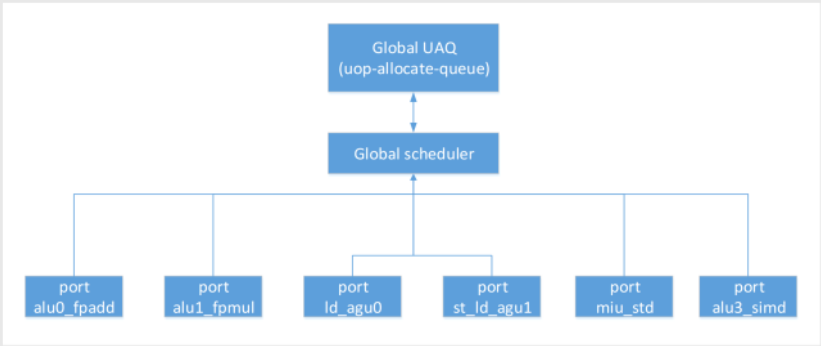
Port binding

2017年12月26日 18:43

Port binding

Port binding实际描述了后端部分(uaq(uop-allocate-queue)+scheduler+port)之间的映射关系，这样在实际硬件组织的时候，可以映射为不同的cluster的组织形式，在CPU1-demo.cfg中的pb.cfg的描述中，给出了一种简化的映射方式

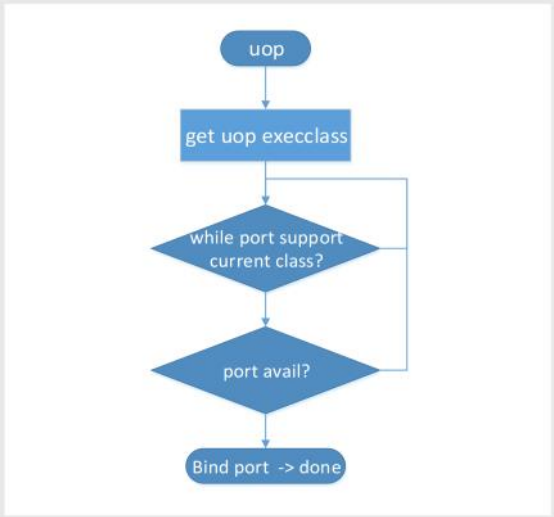


port定义了不同uop实际执行的硬件执行逻辑单元，但是不同uop类型执行的具体时间不是在port上定义的，而是通过DEFUOP进行定义的；同时，DEFEXECSTACK实际上定义了另外一种latency，同时定义了不同stack上的forward latency。对于这个部分，与uop的latency之间的关系还不是很清楚

这个逻辑关系可以这么理解

对于Core而言，只有有限的实际执行逻辑，不同的执行逻辑可以执行不同功能的uop，比如branch和alu操作都是相应的int操作，可以放入到相同的执行单元逻辑中

Porting binding的访问处理流程



Port binding configuration file解析

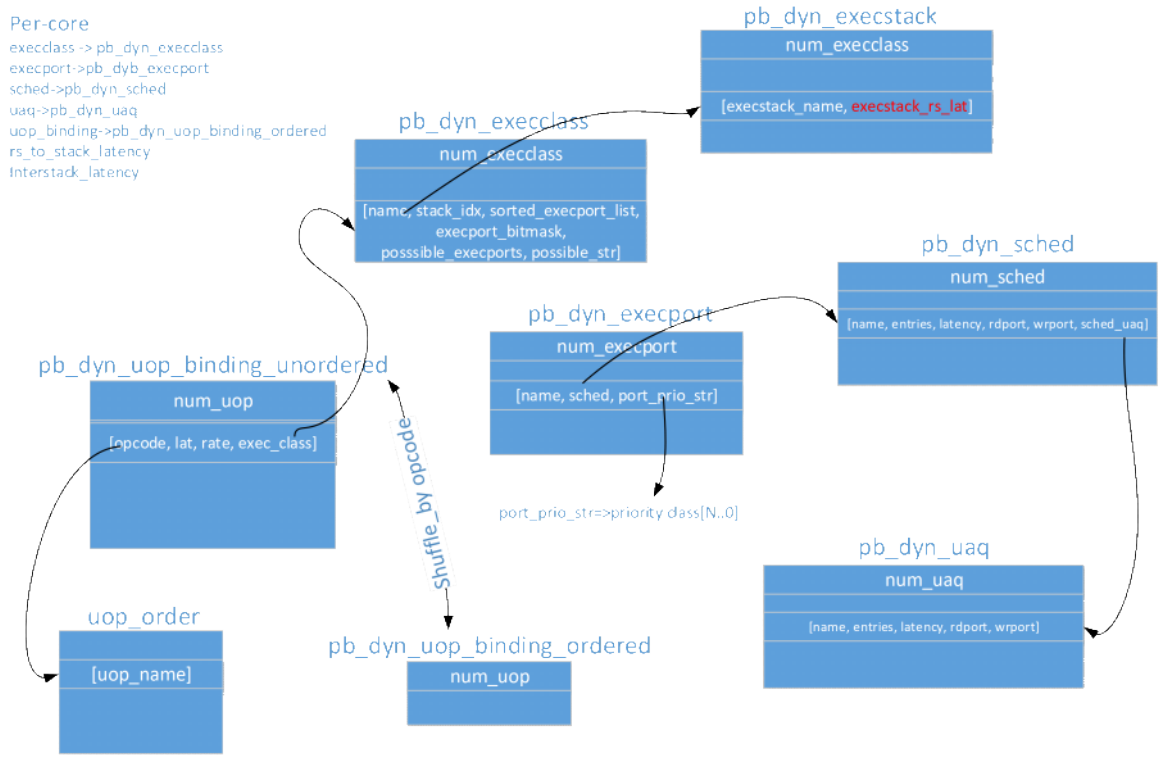
- 原语

原语	影响的变量	含义
DEFEXECPORT(port_id, port_name, exec_port_sched, priority)	pb_dyn_execport	执行的端口定义
DEFEXECCLASS(class_id, class_name, execstack_name)	pb_dyn_execclass	执行的操作类别
DEFSCHED(sched_id, sched_name, sched_entry_num, sched_latency, sched_rdport, sched_wrport, sched_uaq)	pb_dyn_sched	调度器接口，指的哪部分?Reservation station还是instruction queue
DEFUAQ(uaq_id, uaq_name, uaq_entry_num, uaq_latency, uaq_rdport, uaq_wrport)	pb_dyn_uaq	???
DEFUOP(uopname, uop_latency, uop_rate, uop_class_string)	pb_dyn_uop_binding_unordered[uop_idx] pb_dyn_uop_binding_ordered[uop_idx]	uop的类别，操作时延，是否可以pipe?
DEFEXECSTACK(stackname, stack_latency_from_rs)	pb_dyn_execstack_name[execstack_idx] pb_dyn_rs_to_stack_latency[execstack_idx]	表示某种执行功能的部件
DEFINTERSTACK(src_stackname, dst_stackname, inter_latency)	pb_dyn_interstack_latency[src_idx][dst_idx]	定义各个执行功能部件间的传递时延

- 各个部分间的关系

Per-core

pb_dyn_execstack



pb_dyn_execclass

sorted_execport_list：定义了class按照优先级关系(越大越好)，使用的port的list

possible_execbitmask：定义了可用的port的bitmask，port的idx占用1bit

- port相关的管理结构

Execclass[execclass_num]	execclass array
	{
	stack_index 对应的stack
	sorted_execport_list 按照优先级排序的可执行的execport
	possible_execports 可以执行该class的execport个数
	}

Execport[execport_num]	execport array
	{
	sched port对应的scheduler
	}

Sched[scheduler_num]	scheduler array
	{
	entires scheduler的个数
	latency sheduler的调度延时
	read_ports scheduler的读口资源
	write_ports scheduler的写口资源
	uaq scheduler对应的uaq
	}

Uaq[uaq_num]	uop allocate queue array
	{
	entries uaq的个数
	latency uaq的pop 时延？？
	read_ports uaq的读口资源
	write_ports uaq的写口资源
	}

uop_binding	uop info
	{
	opcode uop的opcode
	execclass uop对应的执行类别
	latency uop的执行时延
	rate uop的执行速率
	}

interstack_latency[MAX_EXEC_STACKS] [MAX_EXEC_STACKS]	stack之间的forwarding时延，具体含义要再看代码
rs_to_stack_latency[MAX_EXEC_STACKS]	RS到stack的时延(RS写入到execport的时延？？(dispatch的时延))

execport_free_last_cycle[execport_num]	上一个T后execport的free状态 = !dispatch_port_busy[port][sched] && (!setting_pb_rebind_wb_unused || ! execport_wb_conflict[port])
execport_decaying_counter[execport_num]	根据execport_free_last_cycle的状态决定+/- Execport_free_last_cycle Busy +setting_pb_decay_inc_amount，最大到setting_pb_decay_max_amount Idle -setting_pb_decay_dec_amount，最小到0
execport_ready_counts_last_cycle[execport_num]	上一个T后execport中有多少个uop处于ready状态
execport_wb_conflict[execport_num]	当前port是否出现了write-back bus的冲突，每个T后，自动清0
execport_ready_count[execport_num]	表示当前T有几个uop在execport ready，每个T后，自动清0
execport_real_ready_count[execport_num]	表示当前T有几个uop确实在逻辑上在execport上处于ready状态，每个T后，自动清0
execport_counter[execport_num]	用于进行execport pick时使用的变量，当有uop绑定到execport的时候，按照特定的策略对该counter执行递增操作；当uop从当前execport迁移之后，执行特定策略的递减操作
long_lat_counter[execport_num]	同上一个的作用相同，只不过这个只记录长延时的uop操作(uop_latency>2的)，+1/-1
execclass_counter[execclass_num]	同上一个作用相同，记录的是当前execport上执行的execclass的个数
execclass_uoptags[execlass_num]	只用在round_robin的execport_pick的算法中，每个uoptags是execclass对应的一个标识，表明下一次execclass将会选择的execport的序号(在可选择的execport的范围内)；当某个execclass选择了当前execport之后，其余所有的execclass需要遍历，查看是否他们下一次的选择是否与当前的execport重叠，如果是，那么更换到下一个可选择的execport
cycle_execport_counter[execport_num]	和execport_counter含义相同，这个记录的是上一个T的snapshot，同时在当前T allocate之前；用于在execport_pick的时候根据策略选择execport_counter还是cycle_execport_counter；execport_counter反映的是当前T内execport_pick的实时情况
cycle_long_lat_counter[execport_num]	同上
cycle_execclass_counter[execclass_num]	同上
phytid_execport_counter[MAX_THREADS][execport_num]	同上，从physical thread的角度统计execport的使用情况
phytid_long_lat_counter[MAX_THREADS][execport_num]	同上，从physical thread的角度统计
phytid_execclass_counter[MAX_THREADS][execclass_num]	同上，从physical thread的角度统计
cycle_phytid_execport_counter[MAX_THREADS][execport_num]	同上，记录的每个T之后的情况
cycle_phytid_long_lat_counter[MAX_THREADS][execport_num]	同上，记录的每个T之后的情况
cycle_phytid_execclass_counter[MAX_THREADS][execclass_num]	同上，记录的每个T之后的情况
addingup_execport_counter[execport_num]	单向记录当前execport的uop分配情况，只递增的计数器, monotonic counter
active_branch_colors[MAX_THREADS]	记录当前进行execport分配的uop所属的branch color，实际代表了basic_block的情况(对于跳转的情况) 用在increment_pb_counter和decrement_pb_counter中，如果当前color不相等，不更新counter
last_beuflush_uop_nums[MAX_THREADS]	记录当发生beuflush情况时(branch miss flush)，最新的引起beuflush的uop_num(index) 用在increment_pb_counter和decrement_pb_counter中，如果当前uop older than beuflush uop，不更新counter
bias_matrix[execport_num][execport_num]	当pickup execport失败的时候，设置的portXport的bias值，用于进行port repick的时候对于不同port选择的一个bias选择参数
tmp_execport_loading[execport_num]	只用于pick_ll_class_execport算法，临时变量
tmp_sorted_execport[execport_num]	只用于pick_approx_inline_execport算法，临时变量
tmp_bitmask_count[2^execport_num]	只用于pick_approx_inline_execport算法，每T自动清0，当有新的uop到对应的execport的时候，更新对应的counter计数+1，index为每个execclass可以分配到的execport的位组合
num_sched	有多少个scheduler
num_uaq	有多少个uaq
num_execport	有多少execport
num_execclass	有多少个execclass
num_uop	有多少个uop
num_execstack	有多少个stack
num_interstack	interstack之间的forwarding关系
port_list[classnum][portnum] 只在初始化使用	execlass->execport的优先级定义, 解析execport上面的priority_str，string的格式为class_priority[N…0]，定义了每个port上面对于不同class的优先级关系，0表示某个port不支持某个class。port_list[x]={.priority, .execport}

Port-binding API

clear_pb_counters(phytid, robentry->uop_num)
update_pb_cycle_counters()

Port-binding flow

port的绑定可以发生在allocate stage，也可以在scheduler stage动态进行，目前使用的是allocate stage就进行Port绑定(setting_bind_execport_at_alloc)

pick_execport(CpuframeworkUop *, string *, bool use_bias)

- 根据setting_pb_method选择不同的算法来实现uop bind到不同的execport
  默认使用PB_LEAST_LOADED算法
- 对于enable bias且使用freq override的方式，则再根据已选出的execport使用freq进行再次选择
- 对于lock_load的uop，如果setting_pb_force_lock_load_to_port，那么lock_load uop必须选择对应的port

不同的execport绑定算法

  - PB_ROUND_ROBIN
  - PB_PSEUDO_RANDOM
  - PB_LEAST_LOADED_CLASS
  - PB_FLAT_PRIORITY
  - PB_LEAST_LOADED_LATENCY
  - PB_LEAST_LOADED_DECAYING
  - PB_APPROX_INLINE
  - PB_LEAST_LOAD
    Uop的opcode->execclass->possible_execport(所有可能的execport使用sorted_execport_list进行遍历)
    遍历所有possible_execport，从中选择execport_counter最小的一个(如果有bias，加入bias的值)；如果有多个最小值待选，则通过当前的thread_cycle计算出一个candidate

- 最后，以bind execport的uop增加对应execport的counter

Port-rebinding flow

Check_for_rebind(Uop_Opcode opcode, int execport, uint32 failure_count, uint32 rebind_count, Rebind_Point point)

Execport 当前uop所在的execport

- 根据rebind_point的原因检查是否可以进行uop的rebind
  - RP_NOT_READY
    uop在某个scheduler上的时候，操作数不ready
    如果允许setting_pb_not_ready_rebinding，那么还需要检查当前execport_ready_counts[execport]是否小于threshold(2)
  - RP_WB_CONFLICT
    uop当前所在的execport的write_port有冲突
    如果不允许setting_pb_dont_rebind_on_wb_conflict，那么返回
  - RP_PORT_CONFLICT
    Uop bind的execport没有空间可以容纳当前uop
    - 如果没设置setting_pb_rebind_threshold和setting_pb_hunger_rebind，返回
    - 如果设置了setting_pb_rebind_multiple_times，且当前rebind_count > 0，返回
    - 如果设置了setting_pb_hunger_rebind且execport_ready_counts[execport]<3, 返回
    - 设置了setting_pb_rebind_threshold，且当前failure_count小于该值，返回
- 检查是否可以进行rebind，即尝试一次rebind，看是否可以rebind成功，但是本次不进行真正的rebind

Pick_rebind_execport(Uop_Opcode opcode, int execport, int *attempts)

Execport 当前uop所在的execport

- 从uop当前已bind的execport逐个遍历所有可能的execport
  - !setting_pb_rebinding_only_unused || new_port_is_free(读取变量pb->execport_free_last_cycle)
    &&
  - !setting_pb_rebind_other_port_threshold || new_port_ready_num(读取变量pb->execport_ready_count_last_cycle)
    < setting_pb_rebind_other_port_threshold, return new_port
  - 否则，return invalid_port

Rebind_uop(CpuframeworkUop *s, Pipeline_RecordId plr, int phytid, int color, int64 uop_num, bool rebind_back)

rebind_back 表示某次bind是否是延迟操作，即uop已经重新进行了rebind，但是还没有更新到execport的结构上

重新bind uop的execport到新的execport，通过调用pick_rebind_execport完成，重新bind后，uop的如下字段进行设置

Execport	uop重新bind到的execport
Ctr_execport	uop重新bind后需要update execport计数结构的execport，通常和execport一致
Execport_rebinds	uop重新bind的次数
Execport_sched_failures	uop在当前execport上schedule失败的次数，重置为0