Scheduling Multithreaded Computations By Work-Stealing

Robert D. Blumofe
The University of Texas, Austin

Charles E. Leiserson,
MIT Laboratory for Computer Science

Motivation

- Strict binding of multi-threaded computations on parallel computers.
- To find a parallel execution process by creating a directed acyclic graph and to traverse the instructions accordingly.

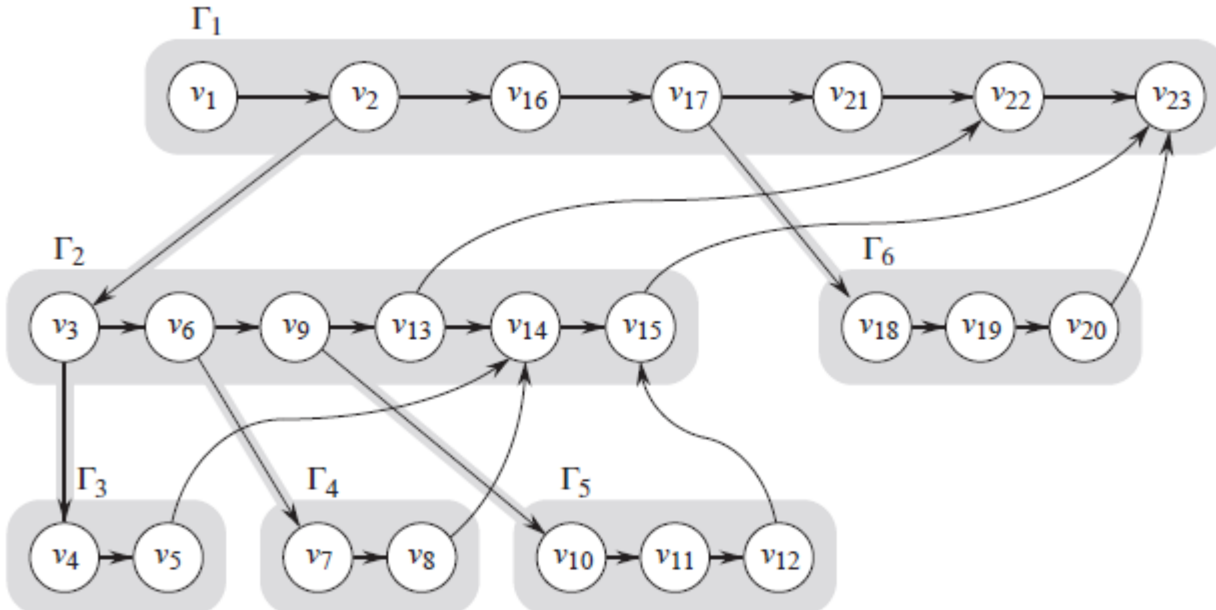
MIMD

- To maintain efficiency, enough threads must be active.
- Number of active threads should be within the hardware requirements.
- Related threads should be placed in the same processor.

Work Sharing vs Work Stealing

- Work Sharing:
  – Scheduler attempts to migrate threads to other processors hoping to distribute work to underutilized processors.
- Work Stealing:
  – Underutilized processors take the initiative by ‘steal’ing threads form other processors.

Multithreaded Computation



Blocks -> Threads
Circles - > Instructions
Right arrows -> Continue Threads
Curved Arrows -> Join Edges
Vertical/Slant arrows -> Spawn Edges

Strict vs Fully-Strict computation

- In a strict computation, all join edges from a thread go to ancestor of the thread in an activation tree.
- In a fully-strict computation all join edges form a thread go to the thread’s parent.

Busy-Leaves

From the time thread T, is spawned until the time T dies, there is atleast one thread from the sub-computation that is ready.

<u>Disadvantages:</u>

- Not efficient in large environment of multiprocessors.
- Scalability

Busy-Leaves conditions

<u>Spawn</u>:

if T_a spawns T_b, T_a returns to thread pool. The processor starts next step with T_b.

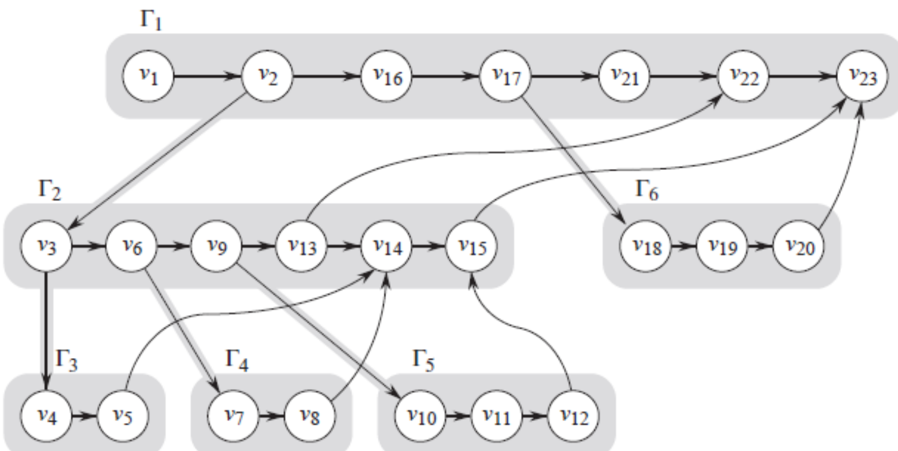
<u>Stall</u>:

if T_a stalls, it is returned to thread pool. Next step by the processor is idle.

<u>Dies</u>:

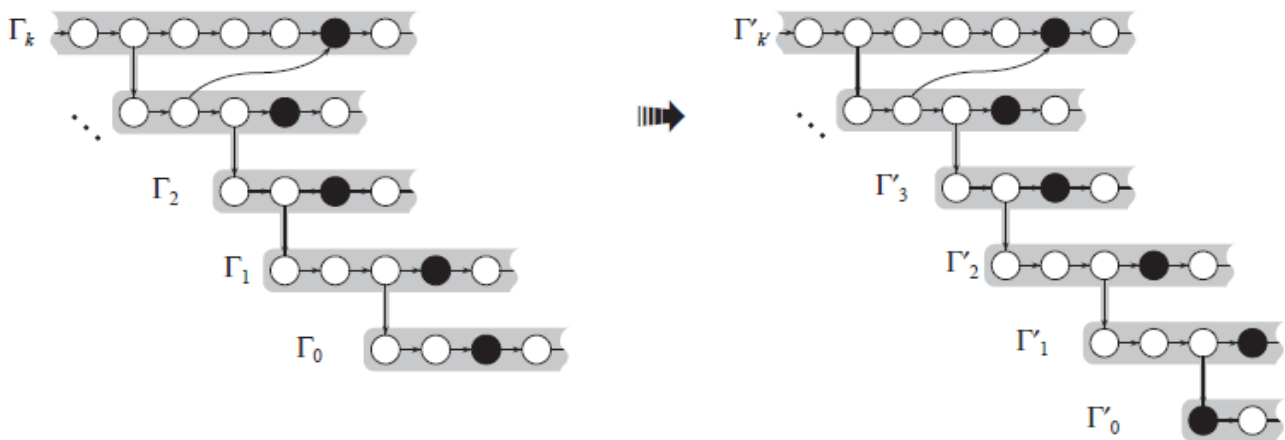
if T_a dies, processor checks if its parent T_b has any living children. If no children and no processor is working on T_b, it is taken from thread pool, else next step idle.

Busy-Leaves conditions

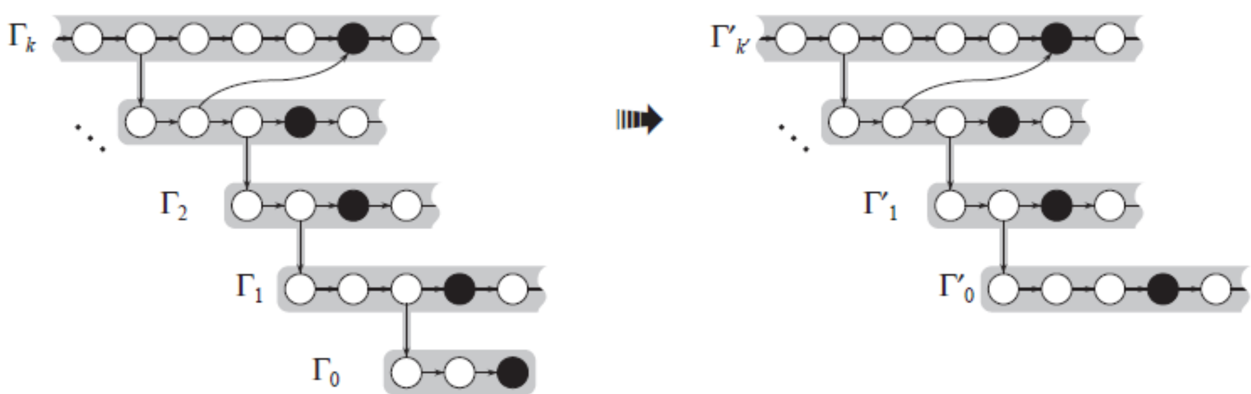


			processor activity			
step	thread pool		p_1		p_2	
1			$\mathbf{\Gamma}_1$:	v_1		
2				v_2		
3			$\mathbf{\Gamma}_2$:	v_3	$\mathbf{\Gamma}_1$:	v_{16}
4		$\mathbf{\Gamma}_2$	$\mathbf{\Gamma}_3$:	v_4		v_{17}
5	$\mathbf{\Gamma}_1$	$\mathbf{\Gamma}_2$		v_5	$\mathbf{\Gamma}_6$:	v_{18}
6	$\mathbf{\Gamma}_1$		$\mathbf{\Gamma}_2$:	v_6		v_{19}
7	$\mathbf{\Gamma}_1$	$\mathbf{\Gamma}_2$	$\mathbf{\Gamma}_4$:	v_7		v_{20}
8		$\mathbf{\Gamma}_2$		v_8	$\mathbf{\Gamma}_1$:	v_{21}
9	Γ_1		$\mathbf{\Gamma}_2$:	v_9		
10	Γ_1		$\mathbf{\Gamma}_5$:	v_{10}	$\mathbf{\Gamma}_2$:	v_{13}
11	$\mathbf{\Gamma}_1$			v_{11}		v_{14}
12		Γ_2		v_{12}	$\mathbf{\Gamma}_1$:	v_{22}
13	Γ_1		$\mathbf{\Gamma}_2$:	v_{15}		
14			$\mathbf{\Gamma}_1$:	v_{23}		

Thread Spawn/Death



(a) Before spawn. (b) After spawn.



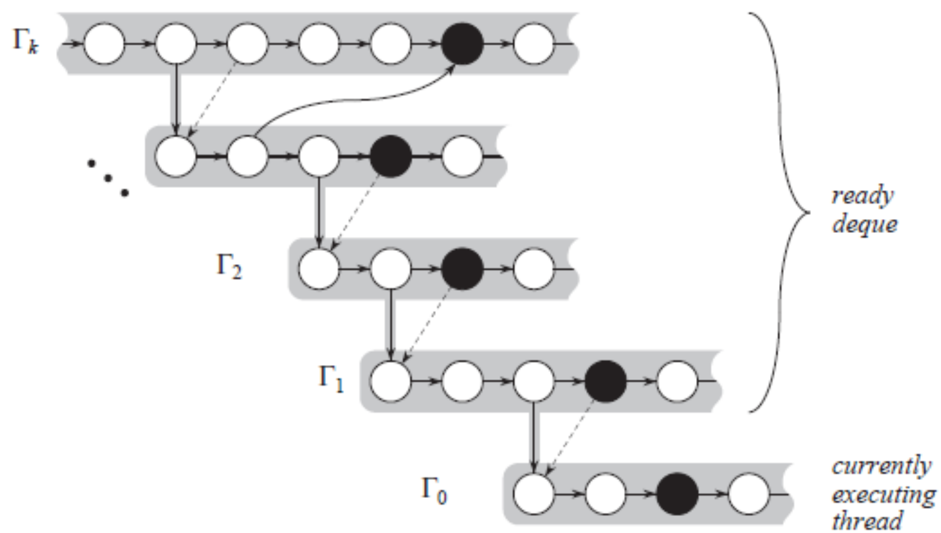
(a) Before death. (b) After death.

Randomized Work-Stealing Algorithm

- The centralized pool of Busy-Leaves algorithm is distributed across the processors.
- Each processor maintains a ready deque.

Cases:

- Spawns
- Stalls
- Dies
- Enables



Randomized Work-Stealing Properties

- For a processor ‘p’, if there are ‘k’ threads in the deque, and if k>0, the following properties are satisfies.

(1) For i=1,2,3,….,k, thread T_i is parent of T_{i-1}

(2) If k>1, for i=1,2,…,k-1, thread T_i has not been worked on since it spawned T_{i-1}

Work-Stealing

- For a fully-strict computation with work 'T_1' and critical path length T_1 the expected running time with P processors is $T_1/P + O(T_1)$
- Execution time on P processors is

  $T_1/P + O(T_1 + \lg P + \lg(1/^{2}))$
- Expected total communication is $O(PT_1(1+n_d)S_{max}$

Recycling Game for Atomic Access

- (P,M) recycling game:

  P = number of balls = number of bins

  M = total number of ball tosses

  – Adversary chooses some of the balls from reservoir selects one of the P bins randomly.
  – Adversary inspects each of the P bins that contains atleast 1 ball and removes it from the bin.

Work Stealing

- Total delay incurred by M random requests made by P processors is $O(M+PlgP+Plog(1/^{2}))$

Conclusion

- The proposed work-stealing scheduling algorithm is efficient in terms of time, space and communication.
- ‘C’ base language called ‘Cilk’ being developed for programming multithreaded computations based on work-stealing.

  http://supertech.lcs.mit.edu/cilk