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 at least one thread from the sub-computation that is ready.

Disadvantages:

• Not efficient in large environment of multiprocessors.

• Scalability
Busy-Leaves conditions

**Spawn:**

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

**Stall:**

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

**Dies:**

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

<table>
<thead>
<tr>
<th>step</th>
<th>thread pool</th>
<th>processor activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\Gamma_1$</td>
<td>$p_1$: $v_1$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>$p_2$: $v_2$</td>
</tr>
<tr>
<td>3</td>
<td>$\Gamma_2$</td>
<td>$\Gamma_1$: $v_3$</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>$\Gamma_2$: $v_4$</td>
</tr>
<tr>
<td>5</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_5$</td>
</tr>
<tr>
<td>6</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_6$</td>
</tr>
<tr>
<td>7</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_7$</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>$\Gamma_2$: $v_8$</td>
</tr>
<tr>
<td>9</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_9$</td>
</tr>
<tr>
<td>10</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_{10}$</td>
</tr>
<tr>
<td>11</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_{11}$</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>$\Gamma_2$: $v_{12}$</td>
</tr>
<tr>
<td>13</td>
<td>$\Gamma_1$</td>
<td>$\Gamma_2$: $v_{13}$</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>$\Gamma_1$: $v_{14}$</td>
</tr>
</tbody>
</table>
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 ‘T₁’ and critical path length T₁ the expected running time with P processors is T₁/P + O(T₁)

• Execution time on P processors is T₁/P + O(T₁ + lg P + lg(1/²))

• Expected total communication is O(PT₁(1+n₃)Sₘₐₓ)
Recycling Game for Atomic Access

- (P,M) recycling game:
  
  \[ P = \text{number of balls} = \text{number of bins} \]
  
  \[ M = \text{total number of ball tosses} \]
  
  - Adversary chooses some of the balls from reservoir selects one of the P bins randomly.
  
  - Adversary inspects each of the P bins that contains at least 1 ball and removes it from the bin.
Work Stealing

- Total delay incurred by $M$ random requests made by $P$ processors is $O(M + P \log P + P \log(1/²))$
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