Types of Schedulers

Thursday, May 19, 2022 6:57 AM

We saw communication ∞ − ∞, ∞ many times, which is done using Condition variable.

Implementations of this are in two examples:

1) Clock (alarm that wakes up threads)
2) Ping Pong (two threads + main thread)

"ball" < [o←] : receives communication
[→ o] : sends communication

x variable is the condition to check for sleep and wait or continue

each thread has one unique lock object
↓
they share the mutex anyway

x = 0 : the thread has no ball ↗ wait for ball
enters waiting and frees mutex
so that other thread checks x
answer : puts back x = 0 after waiting loop
x = 1 : the other thread waits
answer : puts back x = 1 after waiting loop

→ This was an example where threads communicate between each other, many times along time.

→ These are useful when you need threads to wait for other threads to compute something

When we use parallelism, we use the Foster Method:

1) partitioning
2) Communication
3) Union
4) Mapping (Load Balance)
↓
we need scheduler

A scheduler

↳ they work well when $n_{Tasks} < n_{proc}$

otherwise, a processor has many tasks (sequence)

→ Its function is to manage the unbalance of load (when processors differ in load and threads finish first)

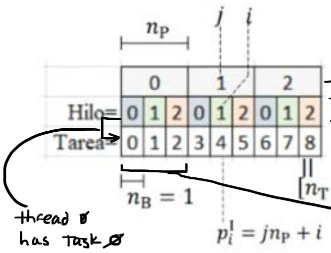
→ Types

1) static
When we know all tasks have the same size
↙
the tasks are assigned
$T_i = P_j$
at the beginning of execution. This is never reassigned.

1) Dynamic
When all tasks have a different size (maybe unknown)

Static: 3 ways of scheduling

1) Cyclical Distribution (One extreme)



	0			1			2		
Hilo=	0	1	2	0	1	2	0	1	2
Tarea=	0	1	2	3	4	5	6	7	8

$p_i^I = jn_P + i$

→ three cycles (equal to np)

-th index (we got 3 threads)

↳ and 9 tasks in total

thread ∅ has Task ∅ and so on (cyclical)

when 3 threads finish these tasks, they take again 3 and execute

j = cycle

i = thread

the position of a Task

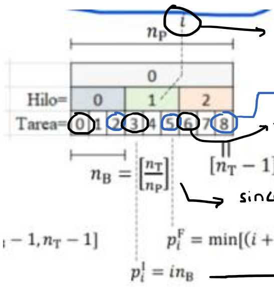
$p_i = j(np) + i$

$p_i = 4$ (id of task)

n_B = num of block

num of Tasks in a processor per cycle

3) Distribution by Block



	0								
Hilo=	0			1			2		
Tarea=	0	1	2	3	4	5	6	7	8

$p_i^F = \min[(i+1)n_B - 1, n_T - 1]$

$p_i^I = in_B$

→ thread (no j since 1 cycle)

2, 5, 8

first tasks for each threads

$$P_i^F = \min[\underbrace{(i+1)_{n_B} - 1}_{\text{first task for next, -1}}, \underbrace{nT - 1}_{\text{last Task}}]$$

↳ since we got 1 cycle

→ the last task for each thread

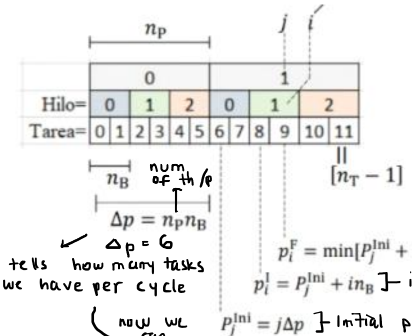
identified by this

↳ $p_1^I = 1(3) = 3$ → for thread 1, its task #1 is 3

→ Here we assume 1 cycle

↳ the bigger the amount of cycles, the smaller n_B

2) Cyclical Distribution by Blocks: Intermediate case



	0						1					
Hilo=	0		1		2		0		1		2	
Tarea=	0	1	2	3	4	5	6	7	8	9	10	11

$p_i^F = \min[P_j^{\text{Ini}} + (i+1)n_B - 1, n_T - 1]$

$p_i^I = P_j^{\text{Ini}} + in_B$

$P_j^{\text{Ini}} = j\Delta p$

num of th/p

$\Delta p = 6$

Δp tells how many tasks we have per cycle

now we can

Final Task for ith thread

→ this term takes care of the case where nTasks is not perfectly divisible by n_B

] initial task for ith thread

] Initial position (task) in jth cycle

↳ the fist task in

→ we have cycles back: say two cycles

↳ threads appear twice

→ Here we assign more Tasks in a thread per cycle

$n_B = 2$ ↳ 2 in this case

a block is more than one task per thread

→ One cycle: becomes type 3)

Many cycles: becomes type 1)] 2) can become this (✓)

→ n_B calculation:

7 tasks

9 threads

$n_B = \frac{7}{3} = \lceil 2.\overline{3} \rceil = 3$

$= \lceil 2.1 \rceil = 3$

$$\boxed{n_B = \left\lceil \frac{nT}{np} \right\rceil}$$

If we assume that task indices are how RAM is organized:

→ the bigger the cycle, the less false sharing

3) avoids false sharing (faster)

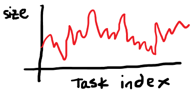
1) May not take care of false sharing (slower)

→ when tasks tends to increase size (or not)

n_B: the bigger n_B, the better behaviour in cache → since False sharing appears

n_B : the bigger n_B, the better behaviour in cache appears
but the worse Load Balance (big differences
when a thread has diff amount of Tasks)
↳ try to have small n_B (for a good Load Balance)
↳ but not too small

2) Dynamic distribution: when the number of sizes
in Tasks is different (or unknown)
↳ or the pattern is very complex
↳ whe use basically a queue
↳ when a processor finishes,
we take the top of the
queue and give the task to
this processor


size
Task index