5008presentation

2023-03-27 5:39 AM

- multithreading algo.
- basics
- motivation : uni-processor
- shared memory
- model to be used

Dynamic Multi-Threading

- In reality it can be difficult to handle multi-threaded programs in a SMP.
- Thus, we will assume a simple concurrency platform that handles all the resources:
  - Scheduler
  - Memory
- It is Called **Dynamic Multi-threading.**

Dynamic Multi-Threading Computing Operations

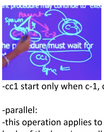
- Spawn
- Sync
- Parallel

Leture 35-Multi threaded merge sort

-spawn:
-when called before a procedure, the parent procedure may continue to execute in parallel

-parent node need not wait for child process, executing parralleling
-parent busy with other process

-sync:
-the keyword sync indicates that the procedure must wait for all its spawned children to complete



-cc1 start only when c-1, c-2, c-3 is finished

-parallel:
-this operation applies to loops, which make possible to execute the body of the loop in parallel

-fib example:

Example
Fibonacci's Definition
$F_0 = 0$
$F_1 = 1$
$F_i = F_{i-1} + F_{i-2}$ for $i > 1$.

Naive Algorithm
fib(n)
if n<2 then return n
x=fib(n-1)
y=fib(n-2)
return x+y

Parallel Algorithm for Fibonacci sequence:
fib(n)
if n<2 then return n;
x=spawn fib(n-1);//parallel execution
y=spawn fib(n-2);//parallel execution
sync;//wait for results of x and y
return x+y;

after find x, and y in parallel

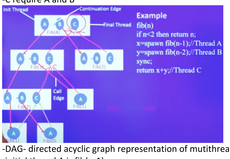
Parallel Algorithm for Fibonacci sequence:
fib(n)
if n<2 then return n;
x=spawn fib(n-1);//parallel execution
y=spawn fib(n-2);//parallel execution
sync;//wait for results of x and y
return x+y;

Multithread computation can be better understood with the help of "Directed Acyclic Graph"(DAG) denoted by G(V,E)
A directed acyclic G = (V, E) graph where the Vertices V are sets of instructions.

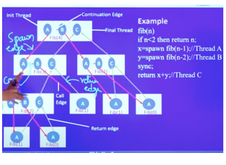
-DAG: show how values are computed, under mutithreading

threadA: x
tB: y
tC: x+y

-C require A and B



-DAG- directed acyclic graph representation of mutithreading
-initial thread A is fib(n-1)
-initial thread B is fib(n-2)
-return x+y is thread C



-edge classification
-continuation edge: connect thread u to successor v within same processor
-spawn edge, return edge

-35
-what is mutithreading merge sort?

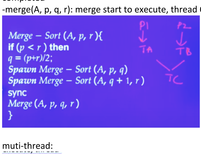
Merge Sort-Serial version

```
Merge – Sort (A, p, r)
{
if (p < r ) then
q = (p+r)/2;
Merge – Sort (A, p, q)
Merge – Sort (A, q + 1, r )
Merge (A, p, q, r )
}
```

Merge Sort-parallel version

```
Merge – Sort (A, p, r ){
if (p < r ) then
q = (p+r)/2;
Spawn Merge – Sort (A, p, q)
Spawn Merge – Sort (A, q + 1, r )
sync
Merge (A, p, q, r )
}
```

-sync: sync start whenever children, after these two process are completed
-merge(A, p, q, r): merge start to execute, thread C



muti-thread:

single thread:

-merge sort 3 parameters: A is list, p is starting, r is ending
-find middle index q
-apply spawn -> thread A
-apply spawn -> thread B
-sync starts whenever the children are completed, after the completion of thread A and B
-merge will be executed
-how parallel?
-previously single processor, list gets divided into process A and B
-whereas now process P1 go to thread A, process P2 go to process B
-combine these two thread using thread C

Analysis

- Recurrence relation of merge sort is:
  $T(n)=2T(n/2)+c.n$
  $=\theta(n\log n)$
- Recurrence relation of Parallel merge sort is:
  $T(n)=T(n/2)+c.n=\theta(n)$