Example of Serial vs Parallel solutions: compute n values and add them together

Wednesday, February 7, 2024 6:46 PM

Example of Serial vs Parallel solutions: compute n values and add them together

- Serial:

```
sum = 0;
for (i = 0; i < n; i++) {
   x = Compute_next_value(. . .);
   sum += x;
}
```

Notes:
- This will simply add to the sum after computing the value in steps starting from i all the way to n
  - STEPS

- Parallel:

```
my_sum = 0;
my_first_i = . . . ;
my_last_i = . . . ;
for (my_i = my_first_i; my_i < my_last_i; my_i++) {
   my_x = Compute_next_value( . . .);
   my_sum += my_x;
}
```

all cores perform this

Each core uses it's own private variables and executes this block of code independently of the other cores.

Notes:
- We have p cores
  - P is much smaller than n
    - Because of this, each core performs a partial sum of approximately n/p values
      - Partial sum meaning: $\sum_{i=0}^{4} i + \sum_{i=5}^{10} + \sum_{i=11}^{17} + \dots$
        - Each summation represents a core
- Explaining the example:
  - Each core starts with a sum of 0 and it handles a certain range from my_first_i to my_last_i and it computes the sum of that range
    - All the cores together will handle up to n

Example (cont.)

- After each core completes execution of the code, is a private variable my_sum contains the sum of the values computed by its calls to Compute_next_value.
- Ex., 8 cores, n = 24, then the calls to Compute_next_value return:

1,4,3, 9,2,8, 5,1,1, 5,2,7, 2,5,0, 4,1,8, 6,5,1, 2,3,9

- Once all the cores are done computing their private my_sum, they form a global sum by sending results to a designated "master" core which adds the final result.

```
if (I'm the master core) {
   sum = my_x;
   for each core other than myself {
      receive value from core;
      sum += value;
   }
} else {
   send my_x to the master;
}
```

Notes:
- The master core will handle adding up all of the sums found from the cores

Example (cont.)

first example

Core	0	1	2	3	4	5	6	7
my_sum	8	19	7	15	7	13	12	14

- Global sum

  8 + 19 + 7 + 15 + 7 + 13 + 12 + 14 = 95

Core	0	1	2	3	4	5	6	7
my_sum	95	19	7	15	7	13	12	14

But wait!

There's a much better way to compute the global sum.

Second example

Better parallel algorithm

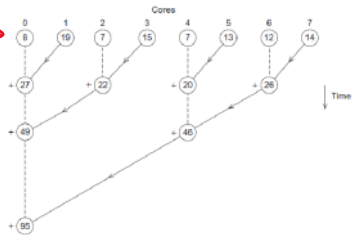
- Don't make the master core do all the work.
- Share it among the other cores.
- Pair the cores so that core 0 adds its result with core 1's result.
- Core 2 adds its result with core 3's result, etc.
- Work with odd and even numbered pairs of cores.

Better parallel algorithm (cont.)

- Repeat the process now with only the evenly ranked cores.
- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.
- Now cores divisible by 4 repeat the process, and so forth, until

Notes:

Multiple cores forming a global sum



- Core 0 adds result from core 2.
- Core 4 adds the result from core 6, etc.

- Now cores divisible by 4 repeat the process, and so forth, until core 0 has the final result.

Analysis

- In the first example, the master core performs 7 receives and 7 additions.

- In the second example, the master core performs 3 receives and 3 additions.

- The improvement is more than a factor of 2!

Notes:

Analysis (cont.)

- The difference is more dramatic with a larger number of cores.
- If we have 1000 cores:
  - The first example would require the master to perform 999 receives and 999 additions.
  - The second example would only require 10 receives and 10 additions.
- That's an improvement of almost a factor of 100!