L5 - S3 Embarrassingly Parallel Computations

Embarrassingly Parallel Computations 易并行运算

- A computation that can be divided into a number of completely independent parts, each of which can be executed by a separate processor.

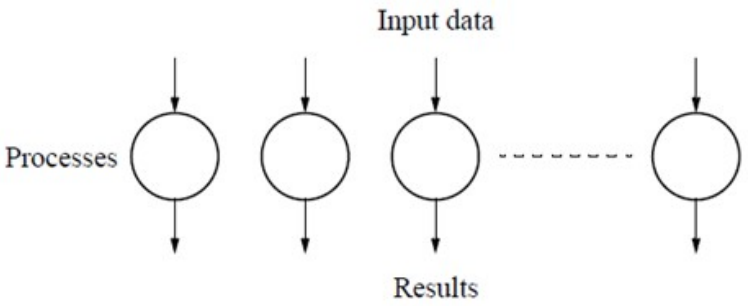

Figure 3.1 Disconnected computational graph (embarrassingly parallel problem).

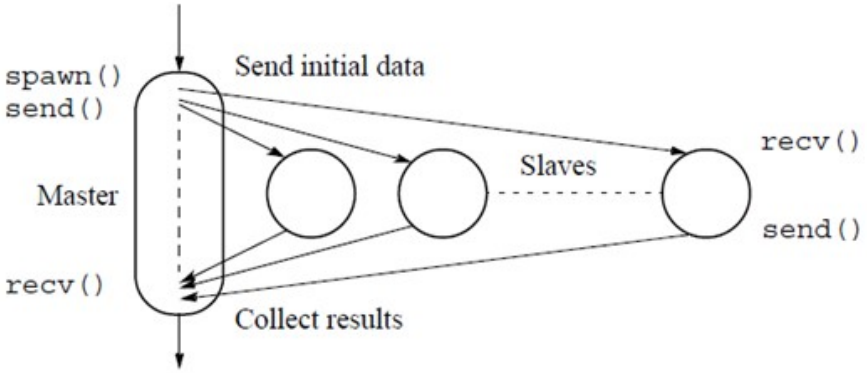

Figure 3.2 Practical embarrassingly parallel computational graph with dynamic process creation and the master-slave approach.

Example - low level image operations

- Shifting Scaling Rotation

- time complexity

Analysis

Sequential

$$t_s = n^2 = O(n^2)$$

Parallel

Communication

$$t_{comm} = t_{startup} + mt_{data}$$

$$t_{comm} = p(t_{startup} + 2t_{data}) + 4n^2(t_{startup} + t_{data}) = O(p + n^2)$$

Computation

$$t_{comp} = 2\left(\frac{n^2}{p}\right) = O(n^2/p)$$

Overall Execution Time

$$t_p = t_{comp} + t_{comm}$$

For constant p, this is $O(n^2)$. However, the constant hidden in the communication part far exceeds those constants in the computation in most practical situations.

Example - Mandelbrot Set Computation

- analysis

Analysis

Sequential

$$t_s \leq \max \times n = O(n)$$

Parallel program

Phase 1: Communication - Row number is sent to each slave

$$t_{comm1} = s(t_{startup} + t_{data})$$

Phase 2: Computation - Slaves perform their Mandelbrot computation in parallel

$$t_{comp} \leq \frac{\max \times n}{s}$$

Phase 3: Communication - Results passed back to master using individual sends

$$t_{comm2} = \frac{n}{s}(t_{startup} + t_{data})$$

Overall

$$t_p \leq \frac{\max \times n}{s} + \left(\frac{n}{s} + s\right)(t_{startup} + t_{data})$$

Example - Monte Carlo Method anyway

- 随机

The ratio of the area of the circle to the square is given by

$$\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$

L6 -S4 Partitioning and Divide-and-Conquer Strategies

Example - Adding a Sequence of Numbers

普通方法

- Dividing sequence into m parts added independently to create partial sums.
- Using separate send() and recv()
  - master 挨个send数据，slave接收、计算、send结果给master，repeat
- Using Broadcast/Multicast Routines
  - master一起bcast，.....，
- Using Scatter and Reduce Routines

Master

```
s = n/m;                                        /* number of numbers */
scatter(numbers,&s,P_group,root=master);        /* send numbers to slaves */
reduce_add(&sum,&s,P_group,root=master);        /* results from slaves */
```

Slave

```
scatter(numbers,&s,P_group,root=master);        /* receive s numbers */
    .                                           /* add numbers */
reduce_add(&part_sum,&s,P_group,root=master);   /* send sum to master */
```

- Analysis

Requires $n - 1$ additions with a time complexity of $O(n)$.

Parallel: Using individual send and receive routines

Phase 1 — Communication

$$t_{comm1} = m(t_{startup} + (n/m)t_{data})$$

Phase 2 — Computation

$$t_{comp1} = n/m - 1$$

Phase 3 — Communication: Returning partial results using send/recv routines

$$t_{comm2} = m(t_{startup} + t_{data})$$

Phase 4 — Computation: Final accumulation

$$t_{comp2} = m - 1$$

Overall

$$t_p = (t_{comm1} + t_{comm2}) + (t_{comp1} + t_{comp2}) = 2mt_{startup} + (n + m)t_{data} + m + n/m - 2$$
$$= O(n + m)$$

Parallel time complexity is worse than sequential time complexity.

Divide and Conquer

- 拆分为subproblem， recursive

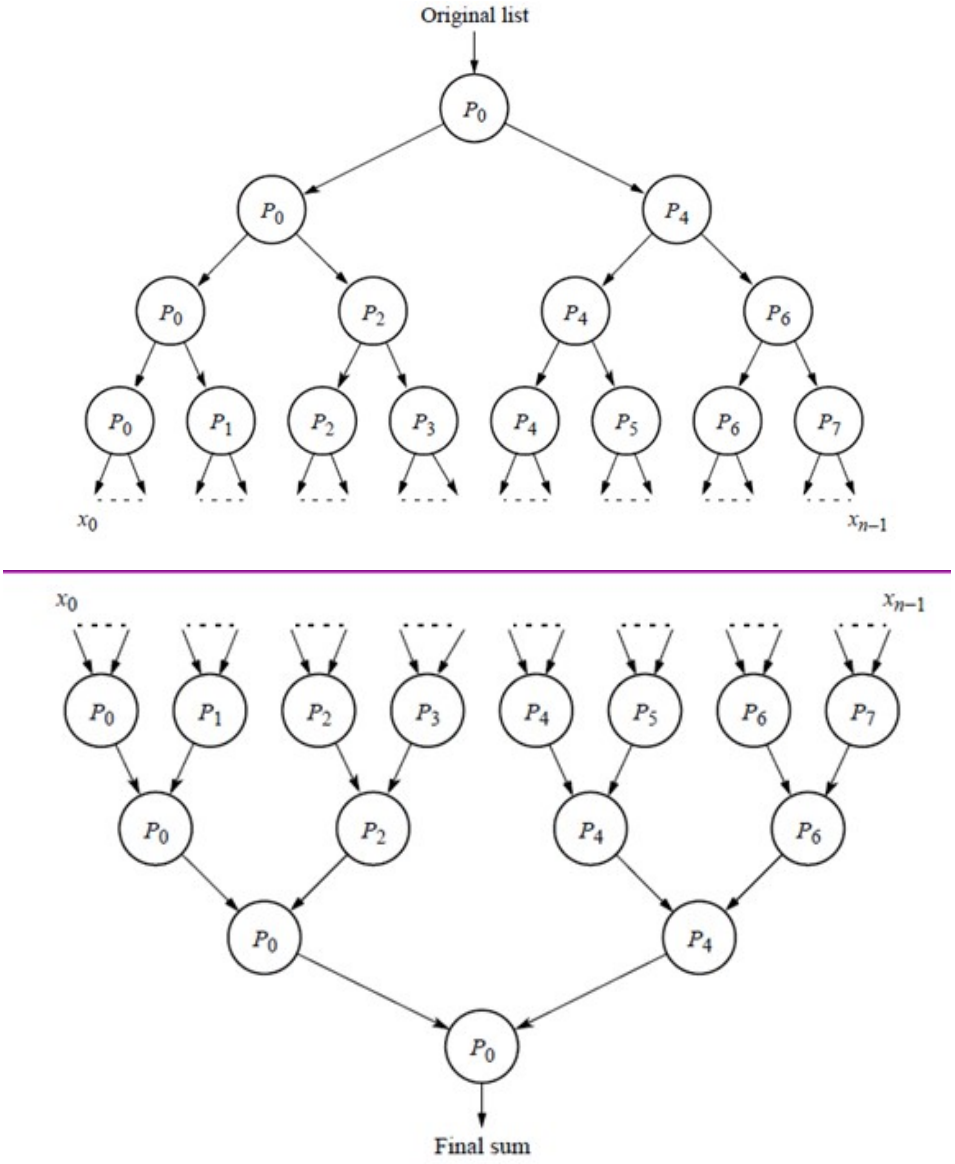

Figure 4.4 Partial summation.

- implementation

Process P_4

```
recv(s1, P0);                  /* division phase */
divide(s1, s1, s2);
send(s2, P6);
divide(s1, s1, s2);
send(s2, P5);
part_sum = *s1;                /* combining phase */
recv(&part_sum1, P5);
part_sum = part_sum + part_sum1;
recv(&part_sum1, P6);
part_sum = part_sum + part_sum1;
send(&part_sum, P0);
```

Similar sequences are required for the other processes.

- Analysis

Analysis

Assume n is a power of 2. Communication setup time, $t_{startup}$, not included.

Communication

Division phase

$$t_{comm1} = \frac{n}{2}t_{data} + \frac{n}{4}t_{data} + \frac{n}{8}t_{data} + \ldots + \frac{n}{p}t_{data} = \frac{n(p-1)}{p}t_{data}$$

Combining phase

$$t_{comm2} = t_{data}\log p$$

Total communication time

$$t_{comm} = t_{comm1} + t_{comm2} = \frac{n(p-1)}{p}t_{data} + t_{data}\log p$$

Computation

$$t_{comp} = \frac{n}{p} + \log p$$

Total Parallel Execution Time

$$t_p = \frac{n(p-1)}{p}t_{data} + t_{data}\log p + \frac{n}{p} + \log p$$

M-array Divide and Conquer

- 之前divide into 2 parts， 现在比如divide into 4 parts

Examples - Sorting Using Bucket Sort

- range of number is divided into *m* equal bucket， 桶内排序，再merge

Sequential time

$$t_s = n + m((n/m)\log(n/m)) = n + n\log(n/m) = O(n\log(n/m))$$

Parallel Algorithm

Bucket sort can be parallelized by assigning one processor for each bucket - reduces second term in the preceding equation to $(n/p)\log(n/p)$ for p processors (where $p = m$).

- 类似于odd-even sort

Further Parallelization

By partitioning the sequence into m regions, one region for each processor.

Each processor maintains p "small" buckets and separates the numbers in its region into its own small buckets.

These small buckets are then "emptied" into the p final buckets for sorting, which requires each processor to send one small bucket to each of the other processors (bucket i to processor i).

- Analysis

Analysis

Phase 1 — Computation and Communication (Partition numbers)

$$t_{comp1} = n$$

$$t_{comm1} = t_{startup} + t_{data}n$$

Phase 2 — Computation (Sort into small buckets)

$$t_{comp2} = n/p$$

Phase 3 — Communication (Send to large buckets)

If all the communications could overlap:

$$t_{comm3} = (p-1)(t_{startup} + (n/p^2)t_{data})$$

Phase 4 — Computation (Sort large buckets)

$$t_{comp4} = (n/p)\log(n/p)$$

Overall

$$t_p = t_{startup} + t_{data}n + n/p + (p-1)(t_{startup} + (n/p^2)t_{data}) + (n/p)\log(n/p)$$

Assumed that numbers are uniformly distributed to obtain these formulas.
Worst-case scenario would occur when all the numbers fell into one bucket!

- "all-to-all" routine could be used for Phase 3 -sends data from each process to every other process
    - 按照顺序接收，P0发给其他的index 0

Numerical Integration

Static Assignment

SPMD pseudocode:

Process P_i

```
if (i == master) {        /* read number of intervals required */
    printf("Enter number of intervals ");
    scanf(%d",&n);
}
bcast(&n, Pgroup);       /* broadcast interval to all processes */
region = (b - a)/p;        /* length of region for each process */
start = a + region * i;    /* starting x coordinate for process */
end = start + region;      /* ending x coordinate for process */
d = (b - a)/n;                          /* size of interval */
area = 0.0;
for (x = start; x < end; x = x + d)
    area = area + f(x) + f(x+d);
area = 0.5 * area * d;
reduce_add(&integral, &area, Pgroup);  /* form sum of areas */
```

A reduce operation is used to add the areas computed by the individual processes.

Can simplify the calculation somewhat by algebraic manipulation (see text).

Gravitational N-Body Problem

Barnes - Hut Algorithm

因为N体问题是一种混沌现象，物体未来的运动状态是没有规律可言的，所以N体模拟只能采用Brute Force的策略。一种提升N体问题模拟算法的速度，一种非常重要的思想就是把相互接近的一组物体近似看成单独的一个物体。对于一组距离足够远的物体，我们可以将它的万有引力作用近似看成是其质心的作用。一组物体的质心是这种物体经过质量加权后的平均位置。更正式地说，如果两个物体的位置分别是(x_1, y_1)和(x_2, y_2)，质量分别为m_1和m_2，那么它们的总质量和总质心(x, y)分别为：

$$m = m_1 + m_2$$

$$x = \frac{x_1 \times m_1 + x_2 \times m_2}{m}$$

$$y = \frac{y_1 \times m_1 + y_2 \times m_2}{m}$$