主要内容

太原理工大学
TAIYUAN UNIVERSITY OF TECHNOLOGY

MPI并行奇偶排序算法实例

岳俊宏
E-mail:yuejunhong@tyut.edu.cn; Tel:18234095983

并行排序算法

并行排序算法 冒泡法串行代码

```
void Bubble_sort(
        int   a[]   /* in/out */,
        int   n     /* in     */) {
    int list_length, i, temp;

    for (list_length = n; list_length >= 2; list_length--)
        for (i = 0; i < list_length-1; i++)
            if (a[i] > a[i+1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
            }

} /* Bubble_sort */
```

排序算法并行化

实例：

假设数组为9 5 7

- ✓ 先进行9和5比较：5 7 9
- ✓ 先进行5和7比较：5 9 7

“比较-交换”的顺序对冒泡排序算法的正确性具有较大的影响。

冒泡排序法不适合并行！

奇偶排序

奇偶交换排序是冒泡排序的一个变种，该算法更适合并行化。

定理：设A是一个拥有n个键值的列表，作为奇偶交换排序算法的输入，那么经过n个阶段后，A能够排好序。

- 偶数阶段:

$$(a[0],a[1]),(a[2],a[3]),(a[4],a[5]),\ldots$$

- 奇数阶段:

$$(a[1],a[2]),(a[3],a[4]),(a[5],a[6]),\ldots.$$

实例

开始：5, 9, 4, 3

偶数阶段，比较(5,9) and (4,3)

得到 5, 9, 3, 4

奇数阶段，比较(9,3)

得到5, 3, 9, 4

偶数阶段，比较(5,3) and (9,4)

getting the list 3, 5, 4, 9

奇数阶段，比较(5,4)

getting the list 3, 4, 5, 9

并行排序算法

奇偶交换排序串行代码

```
void Odd_even_sort(
        int a[]  /* in/out */,
        int n    /* in     */) {
    int phase, i, temp;

    for (phase = 0; phase < n; phase++)
        if (phase % 2 == 0) { /* Even phase */
            for (i = 1; i < n; i += 2)
                if (a[i-1] > a[i]) {
                    temp = a[i];
                    a[i] = a[i-1];
                    a[i-1] = temp;
                }
        } else { /* Odd phase */
            for (i = 1; i < n-1; i += 2)
                if (a[i] > a[i+1]) {
                    temp = a[i];
                    a[i] = a[i+1];
                    a[i+1] = temp;
                }
        }
}  /* Odd_even_sort */
```

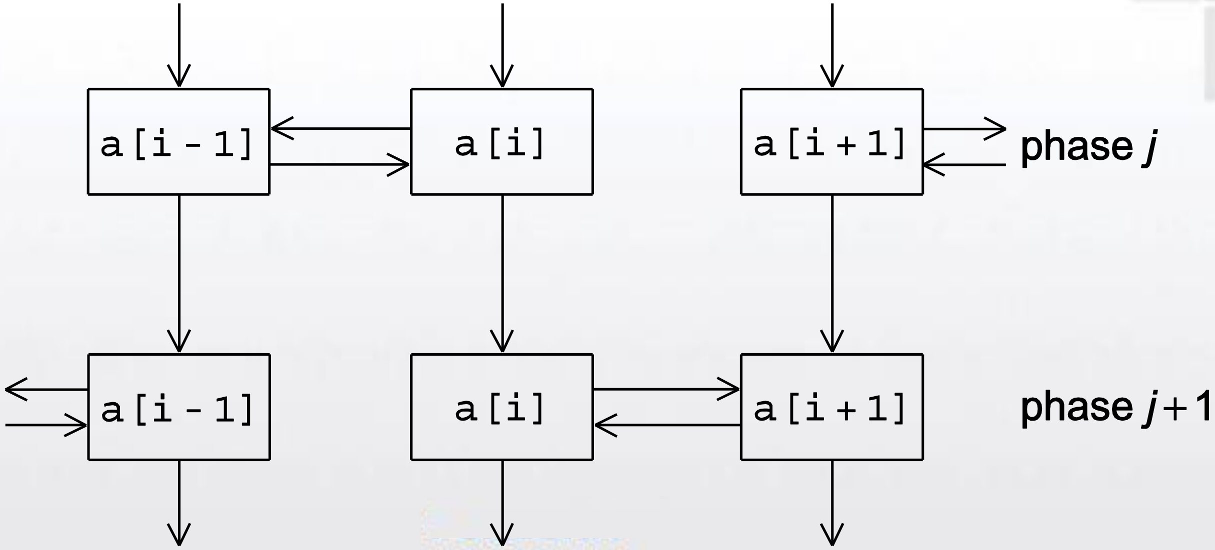
奇偶交换排序并行化

Foster方法：

任务：在阶段j结束时确定a[i]的值

通信：确定a[i]值的任务需要与其他确定a[i+1]或a[i-1]的任务进行通信；

同时，在阶段j结束时，a[i]的值需要用来在阶段j+1结束时确定a[i]的值。



并行排序算法

并行奇偶排序

Time	Process			
	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	9, 11, 15, 16	3, 7, 8, 14	4, 6, 10, 12	1, 2, 5, 13
After Phase 0	3, 7, 8, 9	11, 14, 15, 16	1, 2, 4, 5	6, 10, 12, 13
After Phase 1	3, 7, 8, 9	1, 2, 4, 5	11, 14, 15, 16	6, 10, 12, 13
After Phase 2	1, 2, 3, 4	5, 7, 8, 9	6, 10, 11, 12	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

定理：如果有p个进程运行并行奇偶交换排序算法，则p个阶段后，输入列表排序完毕。

伪代码

```
Sort local keys;
for (phase = 0; phase < comm_sz; phase++) {
   partner = Compute_partner(phase, my_rank);
   if (I'm not idle) {
      Send my keys to partner;
      Receive keys from partner;
      if (my_rank < partner)
         Keep smaller keys;
      else
         Keep larger keys;
   }
}
```

并行排序算法 伪代码

```
if (phase % 2 == 0)          /* Even phase */
   if (my_rank % 2 != 0)         /* Odd rank */
      partner = my_rank - 1;
   else                          /* Even rank */
      partner = my_rank + 1;
else                         /* Odd phase */
   if (my_rank % 2 != 0)         /* Odd rank */
      partner = my_rank + 1;
   else                          /* Even rank */
      partner = my_rank - 1;
if (partner == -1 || partner == comm_sz)
   partner = MPI_PROC_NULL;
```

安全性问题

- 如果进程不是空闲的，可以通过调用MPI_Send和MPI_Recv来实现通信：
  - ✓ MPI_Send(my_keys,n/comm_sz,MPI_INT,partner,0,comm)
  - ✓ MPI_Recv(temp_keys,n/comm_sz,MPI_INT,partner,0,comm,MPI_STATUS_IGNORE);
- 如何评价一个程序是安全的?
- 怎样修改并行奇偶交换排序程序的通信过程，使其安全?

并行排序算法 检验安全

```
int MPI_Ssend(
      void*          msg_buf_p      /* in */,
      int            msg_size       /* in */,
      MPI_Datatype   msg_type       /* in */,
      int            dest           /* in */,
      int            tag            /* in */,
      MPI_Comm       communicator   /* in */);
```

- 输入合适的值和comm_sz，程序没有挂起或者崩溃。

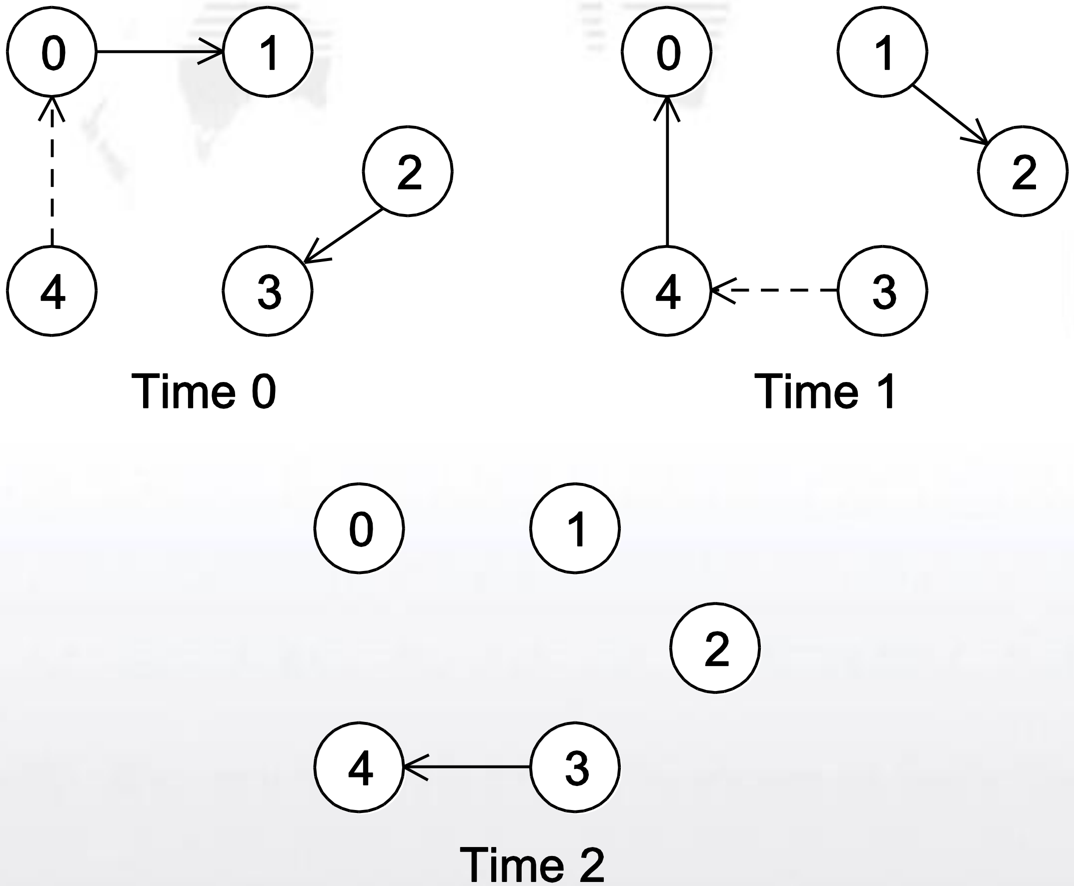
并行排序算法 重构通信

```
MPI_Send(msg, size, MPI_INT, (my_rank+1) % comm_sz, 0, comm);
MPI_Recv(new_msg, size, MPI_INT, (my_rank+comm_sz-1) % comm_sz,
      0, comm, MPI_STATUS_IGNORE.
```

```
if (my_rank % 2 == 0) {
   MPI_Send(msg, size, MPI_INT, (my_rank+1) % comm_sz, 0, comm);
   MPI_Recv(new_msg, size, MPI_INT, (my_rank+comm_sz-1) % comm_sz,
         0, comm, MPI_STATUS_IGNORE.
} else {
   MPI_Recv(new_msg, size, MPI_INT, (my_rank+comm_sz-1) % comm_sz,
         0, comm, MPI_STATUS_IGNORE.
   MPI_Send(msg, size, MPI_INT, (my_rank+1) % comm_sz, 0, comm);
}
```

- comm_sz为偶数和奇数

重构通信



MPI_Sendrecv函数

MPI自己调度通信的方法

```
int MPI_Sendrecv(
      void*          send_buf_p      /* in   */,
      int            send_buf_size   /* in   */,
      MPI_Datatype   send_buf_type   /* in   */,
      int            dest            /* in   */,
      int            send_tag        /* in   */,
      void*          recv_buf_p      /* out  */,
      int            recv_buf_size   /* in   */,
      MPI_Datatype   recv_buf_type   /* in   */,
      int            source          /* in   */,
      int            recv_tag        /* in   */,
      MPI_Comm       communicator    /* in   */,
      MPI_Status*    status_p        /* in   */);
```

MPI_Sendrecv函数

- 调用一次这个函数，分别执行一次阻塞式消息发送和一次消息接收。
- dest和source参数可以相同也可以不同.
- MPI库的这个函数实现了通信调度，使程序不再挂起或崩溃。

```
MPI_Sendrecv(local_A, local_n, MPI_INT, even_partner, 0,
    temp_B, local_n, MPI_INT, even_partner, 0, comm, &status);
```

思考

- 两个进程中的序列如何快速排序交换

9, 11, 15, 16	3, 7, 8, 14
进程0	进程1

```
MPI_Sendrecv(local_A, local_n, MPI_INT, even_partner, 0,
    temp_B, local_n, MPI_INT, even_partner, 0, comm, &status);
```

并行排序算法Merge_low函数

```
void Merge_low(
        int  my_keys[],      /* in/out    */
        int  recv_keys[],    /* in        */
        int  temp_keys[],    /* scratch   */
        int  local_n         /* = n/p, in */) {
    int m_i, r_i, t_i;

    m_i = r_i = t_i = 0;
    while (t_i < local_n) {
        if (my_keys[m_i] <= recv_keys[r_i]) {
            temp_keys[t_i] = my_keys[m_i];
            t_i++; m_i++;
        } else {
            temp_keys[t_i] = recv_keys[r_i];
            t_i++; r_i++;
        }
    }

    memcpy(my_keys, temp_keys, local_n*sizeof(int));
}   /* Merge_low */
```

并行排序算法Merge_high函数

```
void Merge_high(int local_A[], int temp_B[], int temp_C[],
        int local_n) {
   int ai, bi, ci;

   ai = local_n-1;
   bi = local_n-1;
   ci = local_n-1;
   while (ci >= 0) {
      if (local_A[ai] >= temp_B[bi]) {
         temp_C[ci] = local_A[ai];
         ci--; ai--;
      } else {
         temp_C[ci] = temp_B[bi];
         ci--; bi--;
      }
   }

   memcpy(local_A, temp_C, local_n*sizeof(int));
}  /* Merge_high */
```

作业

- 编写奇偶排序串行代码
- 编写奇偶排序并行代码

```c
int main(int argc, char* argv[]) {
   int my_rank, p;
   int *local_A;
   int n,local_n;
   MPI_Comm comm;
   MPI_Init(&argc, &argv);
   comm = MPI_COMM_WORLD;
   MPI_Comm_size(comm, &p);
   MPI_Comm_rank(comm, &my_rank);
   n = 16;//书上例子
   local_n = n/p;
   local_A = (int*) malloc(local_n*sizeof(int));
   Read_list(local_A, local_n, my_rank, p, comm);
   Print_local_lists(local_A, local_n, my_rank, p, comm);
   Sort(local_A, local_n, my_rank, p, comm);
   Print_global_list(local_A, local_n, my_rank, p, comm);
   free(local_A);
   MPI_Finalize();
   return 0;
}  /* main */
```

```
void Sort(int local_A[], int local_n, int my_rank, int p, MPI_Comm comm) {
  int phase;
  int *temp_B, *temp_C;
  int even_partner;  /* phase is even or left-looking */
  int odd_partner;   /* phase is odd or right-looking */
  temp_B = (int*) malloc(local_n*sizeof(int));
  temp_C = (int*) malloc(local_n*sizeof(int));
  if (my_rank % 2 != 0) {
    even_partner = my_rank - 1;
    odd_partner = my_rank + 1;
    if (odd_partner == p) odd_partner = MPI_PROC_NULL;  // Idle during odd phase
  } else {
    even_partner = my_rank + 1;
    if (even_partner == p) even_partner = MPI_PROC_NULL;  // Idle during even phase
    odd_partner = my_rank-1;
  }
  qsort(local_A, local_n, sizeof(int), Compare); /* Sort local list using built-in quick sort */
  for (phase = 0; phase < p; phase++)
    Odd_even_iter(local_A, temp_B, temp_C, local_n, phase, even_partner, odd_partner, my_rank, p, comm);
  free(temp_B);
  free(temp_C);
}  /* Sort */
```

```
void Odd_even_iter(int local_A[], int temp_B[], int temp_C[], int local_n, int phase,
    int even_partner, int odd_partner, int my_rank, int p, MPI_Comm comm) {
  MPI_Status status;
  if (phase % 2 == 0) {
    if (even_partner >= 0) {
      MPI_Sendrecv(local_A, local_n, MPI_INT, even_partner, 0,
            temp_B, local_n, MPI_INT, even_partner, 0, comm, &status);
      if (my_rank % 2 != 0) {Merge_high(local_A, temp_B, temp_C, local_n);}
      else{Merge_low(local_A, temp_B, temp_C, local_n);}
    }
  } else { /* odd phase */
    if (odd_partner >= 0) {
      MPI_Sendrecv(local_A, local_n, MPI_INT, odd_partner, 0,
            temp_B, local_n, MPI_INT, odd_partner, 0, comm, &status);
      if (my_rank % 2 != 0){Merge_low(local_A, temp_B, temp_C, local_n);}
      else{Merge_high(local_A, temp_B, temp_C, local_n);}
    }
  }
}  /* Odd_even_iter */
```

结束!