bfs_omp.cpp 中的 bfs_omp() 函数及其调用部分

```
#include "bfs_common.h"
#include "graph.h"
#include <cstdio>
#include <omp.h>

#define ROOT_NODE_ID 0
#define NOT_VISITED_MARKER -1

void my_vertex_set_clear(vertex_set *list) { list->count = 0; }

void my_vertex_set_init(vertex_set *list, int count) {
  list->max_vertices = count;
  list->vertices = (int *)malloc(sizeof(int) * list->max_vertices);
  my_vertex_set_clear(list);
}

// Take one step of "top-down" BFS.  For each vertex on the frontier,
// follow all outgoing edges, and add all neighboring vertices to the
// new_frontier.
void my_top_down_step(Graph g, vertex_set *frontier, vertex_set *new_frontier,
                      int *distances) {

  for (int i = 0; i < frontier->count; i++) {

    int node = frontier->vertices[i];

    int start_edge = g->outgoing_starts[node];
    int end_edge = (node == g->num_nodes - 1) ? g->num_edges
                                              : g->outgoing_starts[node + 1];

    // attempt to add all neighbors to the new frontier
    for (int neighbor = start_edge; neighbor < end_edge; neighbor++) {
      int outgoing = g->outgoing_edges[neighbor];

      if (distances[outgoing] == NOT_VISITED_MARKER) {
        distances[outgoing] = distances[node] + 1;
        int index = new_frontier->count++;
        new_frontier->vertices[index] = outgoing;
      }
    }
```

```
  }
}

void my_bottom_up_step(Graph g, vertex_set *new_frontier,
                       int *distances, int num_threads, vertex_set *list,
                       double beta, int distance) {
  #pragma omp parallel for schedule(guided)
  for (int i = 0; i < num_threads; ++i) {
    my_vertex_set_clear(list + i);
  }
  #pragma omp parallel for schedule(guided)
  for (int i = 0; i < num_nodes(g); ++i) {
    if (distances[i] !=  NOT_VISITED_MARKER) continue;
    int end = (i == g->num_nodes - 1) ? g->num_edges : g->incoming_starts[i + 1];
    for (int v = g->incoming_starts[i]; v < end; ++v) {
      if (distances[g->incoming_edges[v]] == distance) {
        int id = omp_get_thread_num();
        distances[i] = distance + 1;
        list[id].vertices[list[id].count++] = i;
        break;
      }
    }
  }
  int sum = 0;
  #pragma omp parallel for schedule(guided) reduction(+:sum)
  for (int i = 0; i < num_threads; ++i)
    sum += list[i].count;
  if (sum > 1. * num_nodes(g) / beta) {
    new_frontier->count = sum;
    return;
  }
  for (int i = 0; i < num_threads; ++i) {
    for (int j = 0; j < list[i].count; ++j) {
      new_frontier->vertices[new_frontier->count++] = list[i].vertices[j];
    }
  }
}

void bfs_omp(Graph graph, solution *sol) {
  /** Your code ... */
  int num_threads;
  #pragma omp parallel
  {
    #pragma omp master
    num_threads = omp_get_num_threads();
  }
  double beta = 120;
  vertex_set list1;
```

```
  vertex_set list2;
  vertex_set *list = new vertex_set[num_threads];
  my_vertex_set_init(&list1, graph->num_nodes);
  my_vertex_set_init(&list2, graph->num_nodes);
  #pragma omp parallel for schedule(guided)
  for (int i = 0; i < num_threads; ++i)
    my_vertex_set_init(list + i, graph->num_nodes);

  vertex_set *frontier = &list1;
  vertex_set *new_frontier = &list2;

  // initialize all nodes to NOT_VISITED
  #pragma omp parallel for schedule(guided)
  for (int i = 0; i < graph->num_nodes; i++)
    sol->distances[i] = NOT_VISITED_MARKER;

  // setup frontier with the root node
  frontier->vertices[frontier->count++] = ROOT_NODE_ID;
  sol->distances[ROOT_NODE_ID] = 0;

  int distance = 0;

  while (frontier->count != 0) {
    my_vertex_set_clear(new_frontier);
    if (frontier->count > 1. * num_nodes(graph) / beta)
      my_bottom_up_step(graph, new_frontier, sol->distances, num_threads, list,
beta, distance);
    else
      my_top_down_step(graph, frontier, new_frontier, sol->distances);

    // swap pointers
    vertex_set *tmp = frontier;
    frontier = new_frontier;
    new_frontier = tmp;
    ++distance;
  }
}

```

实现思路为：

1. 首先实现 Bottom Up 方式的 BFS，即枚举所有结点，看是否是当前 `frontier` 集合中结点的邻居并且没有访问。
2. 对于结点的枚举采用 OpenMP 加速，即对于最外层循环使用 `#pragma omp parallel for`，而在寻找到可行的结点时，加入线程对应编号的 `list` 中以减少同步；而由于不同的不同的结点分配给了不同的 OpenMP 线程，因此不会出现 `distances` 和 `new_frontier` 的写入竞争。
3. 设置 `beta` 参数，当 `frontier` 中的节点数超过 $\frac{n}{\beta}$ 时，再采用 Bottom Up 方式加速，否则使用单线程的 Top Down 方式进行朴素计算。

bfs_omp_mpi.cpp 中的 bfs_omp_mpi() 函数及其调用部分

```
#include "bfs_common.h"
#include "graph.h"
#include <cstdio>
#include <sys/time.h>
#include <mpi.h>
#include <omp.h>

#define ROOT_NODE_ID 0
#define NOT_VISITED_MARKER -1

void my_vertex_set_clear(vertex_set *list) { list->count = 0; }

void my_vertex_set_init(vertex_set *list, int count) {
  list->max_vertices = count;
  list->vertices = (int *)malloc(sizeof(int) * list->max_vertices);
  my_vertex_set_clear(list);
}

// Take one step of "top-down" BFS.  For each vertex on the frontier,
// follow all outgoing edges, and add all neighboring vertices to the
// new_frontier.
void my_top_down_step(Graph g, vertex_set *frontier, vertex_set *new_frontier,
                      int *distances, int rank, int beta) {
  int distance = distances[frontier->vertices[0]];

  if (rank == 0) {
    for (int i = 0; i < frontier->count; i++) {

      int node = frontier->vertices[i];

      int start_edge = g->outgoing_starts[node];
      int end_edge = (node == g->num_nodes - 1) ? g->num_edges
                                                : g->outgoing_starts[node + 1];

      // attempt to add all neighbors to the new frontier
      for (int neighbor = start_edge; neighbor < end_edge; neighbor++) {
        int outgoing = g->outgoing_edges[neighbor];
        if (distances[outgoing] == NOT_VISITED_MARKER) {
          distances[outgoing] = distances[node] + 1;
          int index = new_frontier->count++;
          new_frontier->vertices[index] = outgoing;
        }
      }
    }
  }
```

```
  MPI_Bcast(&new_frontier->count, 1, MPI_INT, 0, MPI_COMM_WORLD);
  if (new_frontier->count <= 1. * num_nodes(g) / beta) {
    MPI_Bcast(new_frontier->vertices, new_frontier->count, MPI_INT, 0,
MPI_COMM_WORLD);
    #pragma omp parallel for schedule(guided)
    for (int i = 0; i < new_frontier->count; ++i)
      distances[new_frontier->vertices[i]] = distance + 1;
  } else {
    MPI_Bcast(new_frontier->vertices, new_frontier->count, MPI_INT, 0,
MPI_COMM_WORLD);
    #pragma omp parallel for schedule(guided)
    for (int i = 0; i < new_frontier->count; ++i)
      distances[new_frontier->vertices[i]] = distance + 1;
    // MPI_Bcast(distances, num_nodes(g), MPI_INT, 0, MPI_COMM_WORLD);
  }
}

void my_bottom_up_step(Graph g, vertex_set *new_frontier, vertex_set *tmp_frontier,
                       int *distances, int num_threads, vertex_set *list,
                       int rank, int nprocs, int *count,
                       int *disp, double beta, int distance) {
  // timeval start, end;
  // gettimeofday(&start, NULL);

  int blockSize = num_nodes(g) / nprocs;
  if (num_nodes(g) % nprocs) ++blockSize;
  int l = std::min(num_nodes(g), blockSize * rank), r = std::min(num_nodes(g) - 1,
blockSize * (rank + 1));
  #pragma omp parallel for schedule(guided)
  for (int i = 0; i < num_threads; ++i) {
    my_vertex_set_clear(list + i);
  }
  #pragma omp parallel for schedule(guided)
  for (int i = l; i < r; ++i) {
    if (distances[i] !=  NOT_VISITED_MARKER) continue;

    for (int *v = g->incoming_edges + g->incoming_starts[i]; v < g->incoming_edges
+ g->incoming_starts[i + 1]; ++v) {
      if (distances[*v] == distance) {
        int id = omp_get_thread_num();
        distances[i] = distance + 1;
        list[id].vertices[list[id].count++] = i;
        break;
      }
    }
  }
  if ((num_nodes(g) - 1) / blockSize == rank && distances[num_nodes(g) - 1] ==
NOT_VISITED_MARKER) {
```

```
    int i = num_nodes(g) - 1;
    for (int v = g->incoming_starts[i]; v < g->num_edges; ++v) {
      if (distances[g->incoming_edges[v]] == distance) {
        distances[i] = distance + 1;
        list[0].vertices[list[0].count++] = i;
        break;
      }
    }
  }
  int sum = 0;
  #pragma omp parallel for schedule(guided) reduction(+:sum)
  for (int i = 0; i < num_threads; ++i)
    sum += list[i].count;
  // gettimeofday(&end, NULL);
  // printf("Full Rank %d: %lf ms.\n",
  //         rank, (1000000.0 * (end.tv_sec - start.tv_sec) + end.tv_usec -
start.tv_usec) / 1000.0);
  MPI_Allreduce(&sum, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
  if (sum > 1. * num_nodes(g) / beta) {
    new_frontier->count = sum;
    MPI_Allgather(distances + l, blockSize, MPI_INT, distances, blockSize, MPI_INT,
MPI_COMM_WORLD);
    return;
  }
  my_vertex_set_clear(tmp_frontier);
  for (int i = 0; i < num_threads; ++i) {
    for (int j = 0; j < list[i].count; ++j) {
      tmp_frontier->vertices[tmp_frontier->count++] = list[i].vertices[j];
    }
  }
  MPI_Allgather(&tmp_frontier->count, 1, MPI_INT, count, 1, MPI_INT,
MPI_COMM_WORLD);
  for (int i = 0; i < nprocs; ++i) {
    if (i) disp[i] = disp[i - 1] + count[i - 1];
    else disp[i] = 0;
  }
  MPI_Allgatherv(tmp_frontier->vertices, tmp_frontier->count, MPI_INT,
new_frontier->vertices, count, disp, MPI_INT, MPI_COMM_WORLD);
  new_frontier->count = disp[nprocs - 1] + count[nprocs - 1];
  #pragma omp parallel for schedule(guided)
  for (int i = 0; i < new_frontier->count; ++i)
    distances[new_frontier->vertices[i]] = distance + 1;
}

void bfs_omp_mpi(Graph graph, solution *sol) {
  /** Your code ... */
  int rank, nprocs;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
133   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
134
135   int blockSize = num_nodes(graph) / nprocs;
136   if (num_nodes(graph) % nprocs) ++blockSize;
137
138   int *count = new int[nprocs];
139   int *disp = new int[nprocs];
140   int num_threads;
141   #pragma omp parallel
142   {
143     #pragma omp master
144     num_threads = omp_get_num_threads();
145   }
146   double beta = 120;
147   vertex_set list1;
148   vertex_set list2;
149   vertex_set list3;
150   vertex_set *list = new vertex_set[num_threads];
151   my_vertex_set_init(&list1, graph->num_nodes);
152   my_vertex_set_init(&list2, graph->num_nodes);
153   my_vertex_set_init(&list3, blockSize);
154   #pragma omp parallel for schedule(guided)
155   for (int i = 0; i < num_threads; ++i)
156     my_vertex_set_init(list + i, graph->num_nodes);
157
158   vertex_set *frontier = &list1;
159   vertex_set *new_frontier = &list2;
160   vertex_set *tmp_frontier = &list3;
161
162   int *my_distances = new int[nprocs * blockSize];
163   // initialize all nodes to NOT_VISITED
164   #pragma omp parallel for schedule(guided)
165   for (int i = 0; i < graph->num_nodes; i++)
166     my_distances[i] = NOT_VISITED_MARKER;
167
168   // setup frontier with the root node
169   frontier->vertices[frontier->count++] = ROOT_NODE_ID;
170   my_distances[ROOT_NODE_ID] = 0;
171
172   int distance = 0;
173
174   // printf("Rank %d Start!!!!!\n", rank);
175
176   while (frontier->count != 0) {
177     my_vertex_set_clear(new_frontier);
178
179     // timeval start, end;
180     // gettimeofday(&start, NULL);
```

```
181
182     if (frontier->count > 1. * num_nodes(graph) / beta)
183       // printf("Bottom up "),
184       my_bottom_up_step(graph, new_frontier, tmp_frontier,  my_distances,
    num_threads, list, rank, nprocs, count, disp, beta, distance);
185     else
186       // printf("Top down "),
187       my_top_down_step(graph, frontier, new_frontier, my_distances, rank, beta);
188
189     // gettimeofday(&end, NULL);
190     // printf("Rank %d: %lf ms. %d\n",
191     //         rank, (1000000.0 * (end.tv_sec - start.tv_sec) + end.tv_usec -
    start.tv_usec) / 1000.0, new_frontier->count);
192
193     // swap pointers
194     vertex_set *tmp = frontier;
195     frontier = new_frontier;
196     new_frontier = tmp;
197     ++distance;
198   }
199   #pragma omp parallel for schedule(guided)
200   for (int i = 0; i < graph->num_nodes; i++)
201     sol->distances[i] = my_distances[i];
202   // printf("Rank %d End!!!!!\n", rank);
203 }
204
```

实现思路为：

1. 与 `bfs_omp()` 大体一样，但在 Bottom Up 的实现中，预分配给每个进程固定的结点编号序列用以更新。
2. 对于每个进程，仍然采用 OpenMP 加速并合并。
3. 在 Bottom Up 方式结束时，通过 `MPI_Allgather()` 和 `MPI_Allgatherv()` 进行通讯，更新本次迭代的 `new_frontier` 以及 `distances` 数组。
4. 在 Top Down 方式结束时，通过 `MPI_Bcast()` 进行对应信息的广播。

性能优化与效果

1. 通过设置阈值 `beta`（β）来对于不同情况选择 Top Down 与 Bottom Up 两种方式。在 OpenMP 版本的测试中，可以获得 30 ~ 40% 左右的加速。
2. 通过调整 `beta` 与 `num_threads` 的数值。可以在 OpenMP+MPI 版本中获得 20% 左右的加速。
3. 在 Bottom Up 方式结束前，首先判断是否 `new_frontier` 的总点数是否达到了设定的阈值要求即 $\frac{n}{\beta}$，如果达到则不进行线程/进程之间 `new_frontier` 数组的合并，而只是记录总点数（MPI 通讯时通过 `MPI_Allgather()` 同步 `distances` 数组）。在 `graph/68m.graph` 数据集下，两种实现大致均可获得 $10ms$ 左右加速。

OpenMP 不同线程数运行结果

在 `graph/200m.graph` 测试集下，采用 28 线程数，耗时为 $297.3816ms$。

> 注：下面测试中时间均为 `graph/500m.graph` 用时。

线程数	运行时间(ms)	加速比
1	10595.4758	1.00
7	2349.3291	4.51
14	1244.3149	8.52
28	774.3301	13.68

OpenMP+MPI 不同进程数运行结果

在 `graph/500m.graph` 测试集下，采用 4×1 进程数，耗时为 $360.2648ms$。

在 `graph/200m.graph` 测试集下，采用 4×1 进程数，耗时为 $239.1775ms$。

> 注：下面测试中时间均为 `graph/68m.graph` 用时。

机器数 × 进程数	运行时间(ms)	加速比
1×1	46.4430	1.00
1×2	45.4541	1.02
1×4	77.4105	0.60
1×14	95.2084	0.49
1×28	121.5943	0.38
2×1	39.3658	1.18
2×2	52.8389	0.88
2×4	57.9481	0.80
2×14	89.0764	0.52
2×28	126.0286	0.37
4×1	32.8034	1.42
4×2	44.6022	1.04
4×4	57.0379	0.81
4×14	94.0943	0.49
4×28	322.9891	0.14