```
Data  : M, POsToMove
Result: migrate partition objects in POsToMove
begin
    /* STEP 1: collect entities to process and clear partitioning data. See
    §4.4.1 */
    for each M_i^d ∈ POsToMove do
        insert M_i^d into entitiesToUpdate[d];
        reset partition classification and 𝒫;
        for each M_j^q ∈ {∂(M_i^d)} do
            insert M_j^q into entitiesToUpdate[q];
            reset partition classification and 𝒫;
        endfor
    endfor
    /* STEP 2: determine residence partition. See §4.4.2 */
    M_setResidencePartition(POsToMove, entitiesToUpdate[q]);
    /* STEP 3: update partition classification and collect entities to remove.
    See §4.4.3 */
    for d ← 3 to 0 do
        for each M_i^d ∈ entitiesToUpdate[d] do
            determine partition classification;
            if P_local ∉ 𝒫[M_i^d]
                insert M_i^d into entitiesToRemove[d];
            endif
        endfor
    endfor
    /* STEP 4: exchange entities. See §4.4.4 */
    for d ← 0 to 3 do
        M_exchangeEnts(entitiesToUpdate[d]);
    endfor
    /* STEP 5: remove unnecessary entities. See §4.4.5 */
    for d ← 3 to 0 do
        for each M_i^d ∈ entitiesToRemove[d] do
            if M_i^d is on partition boundary
                remove copies of M_i^d on other partitions;
            endif
            remove M_i^d;
        endfor
    endfor
    /* STEP 6: update ownership. See §4.4.6 */
    for each P_i^d in P do
        owning partition of P_i^d ← the poorest partition among 𝒫[P_i^d];
    endfor
end
```

Algorithm 4.2: M_migrate(M, $POsToMove$)

Note: term 'partition' in Algorithm 4.2 means 'partition' or 'part' in ITAPS parallel interface terms.

Algorithm Input Data:

- **M**, a distributed mesh with partition model **P** stored in each part. Entity's partition classification and remote copies are pre-computed.
- **POsToMove**, a list of partition objects to migrate and their destination part ids.

Result: Migrate partition objects in *POsToMove*

- **STEP 1**: Collect entities to process and clear their partition classifications.

  In **STEP 1**, the entities collected in *entitiesToUpdate* include:

  - $M_i^d \in$ *entitiesToUpdate*;
  - For each M_i^d, all downward entities that bound M_i^d, and their remote copies on other parts.

- **STEP 2**: determine residence parts where each mesh entity in *entitiesToUpdate* will exist after migration.
- **STEP 3**: update partition classification for each $M_i^d \in$ *entitiesToUpdate*. For each M_i^d, if its residence parts does not include its current local part, collect it into *entitiesToRemove*.
- **STEP 4**: exchange entities in *entitiesToUpdate* from dimension 0 to 3, send and receive messages (entity information) between parts, and then create entities on the destination parts. The remote copies of the entities created on the destination parts are updated through communication.
- **STEP 5**: remove unnecessary entities in *entitiesToRemove* from dimension 3 to 0. If a mesh entity to remove is on the part boundary, its remote copies on other parts must be removed through communication.
- **STEP 6**: update the ownership for each partition model entity P_i^d stored in the partition model **P**, based on poor-to-rich ownership rule.