## RefineLB

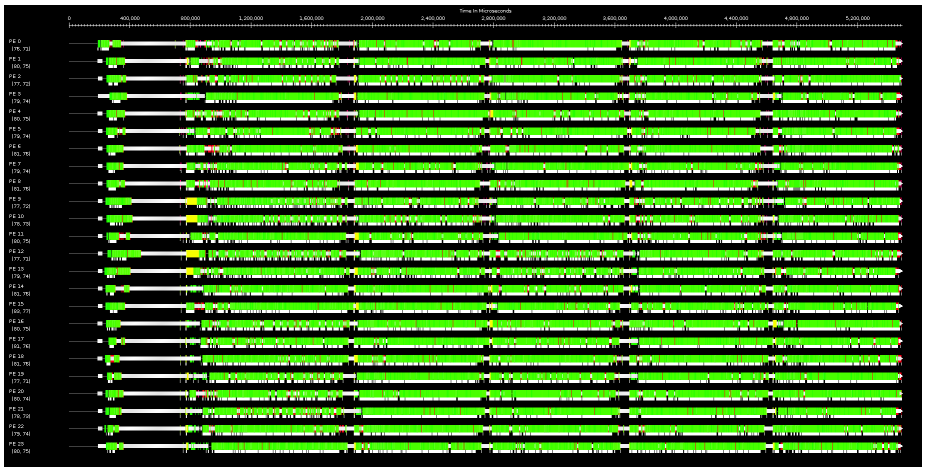

Figure 1: RefineLB: Cores: 24, LB_PERIOD: 20, Iteration Count:100

**Are they beneficial? Why or why not? How much is the overhead?**
Yes, they are beneficial if done less frequently. With LB_Period= 10, RefineLB total execution time (6683 ms) is more that that of without any LB case (6070 ms). But with LB_Period= 20 we are getting better performance (RefineLB time = 5492 ms and NoLB time = 5955 ms). The reason is that with frequent load balancing the overhead of load balancing (decision time and actual migration time) become dominant than the actual gain obtained by load balancing. For example, with LB_Period= 10, the load balancing overhead is 1.441% as compared to .860% at LB_Period= 20.

**Other Overheads**: When we are doing RefineLB, the neighboring chares **MAY** get separated if they where situated at the same PE during initial placement. This observation is true for any LBs discussed in this report ( except the GreedyComm which takes into account the associated communication). This also add on to communication load on CPU.

Also the application is peculiar in two ways: (1) The load is frequently changing across the chares. (2) With too big iteration count the application may load balance itself naturally. Now if we keep the iteration count be very large (e.g. with 500) then the RefineLB is not giving better performance (even with infrequent load balancing), because the application is naturally balanced and the load balancing effort comes as an extra overhead.

**Which strategy is the best for this Particle application?**
This strategy seems to be the best among the three discussed, with the constrains like LB period should not be too frequent and the iteration count should not be too big. The reason are: (1) The overhead of RefineLB is lower than others. (2) As the load is frequently changes in the application it does not worth using a high overhead load balancing strategy. Rather a small refinement will be sufficient.

# GreedyLB

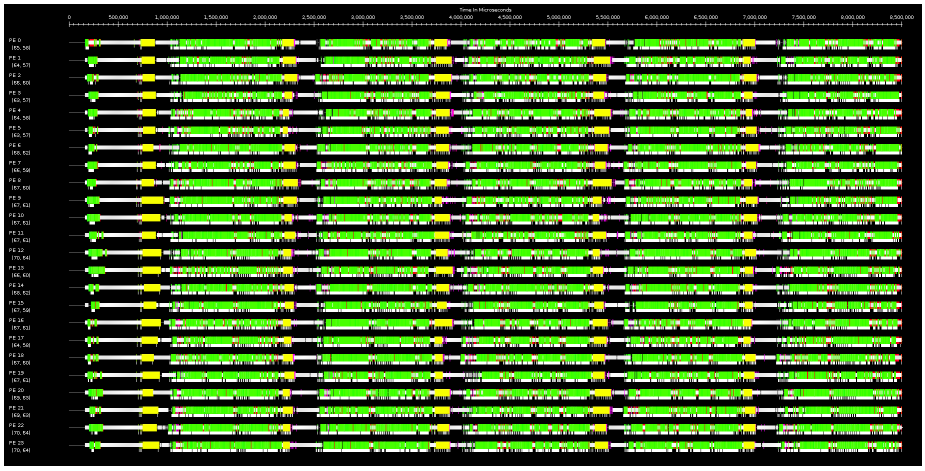

Figure 2: GreedyLB: Cores: 24, LB_PERIOD: 20, Iteration Count:100

**Are they beneficial? Why or why not? How much is the overhead?** NOT beneficial. The following table with different LB_Period values shows that GreedyLB's total execution times are more than those with NoLB and RefineLB.

| LB_Period | $t_{nolb}(ms)$ | $t_{rlb}(ms)(\%p_{rlb})$ | $t_{glb}(ms)(\%p_{glb})$ |
|---|---|---|---|
| 10 | 6070 | 6683 (1.441) | 9830 (11.5) |
| 20 | 5955 | 5492 (.860) | 8503 (7.112) |
| 30 | 5876 | 5277 (.60) | 5989 (5.51) |

Table 1: $t_{nolb}$, $t_{rlb}$, $t_{glb}$ are the executions times of NoLB, RefineLB and GreedyLB respectively and $p_{rlb}$, $p_{glb}$ are the % CPU utilization for the migrations.

The reason is that the overhead of GreedyLB load balancing (decision time and actual migration time) is very high. Though the quality of load balancing provided by GreedyLB is much better than RefineLB, but that does not help in this scenario because (1) The load is frequently changing across the chares. So the effort put in by this expensive load balancing is kind of nullified after few iterations. (2) In case of GreedyLB, the number of chares migrated in more as compared to RefineLB and that migration happens without taking communication into account, so this ends up with decoupling the neighboring chares which in turn causes huge communication load on CPU.

Another interesting observation is as the LB_Period decreases, the overhead of load balancing diminishes and so the total execution time. Also with higher iteration count, the performance is bad as compared to both NoLB and RefineLB because the application get naturally load balanced in that scenario and the effort of this expensive load balancing becomes an overhead.

# GreedyCommPB

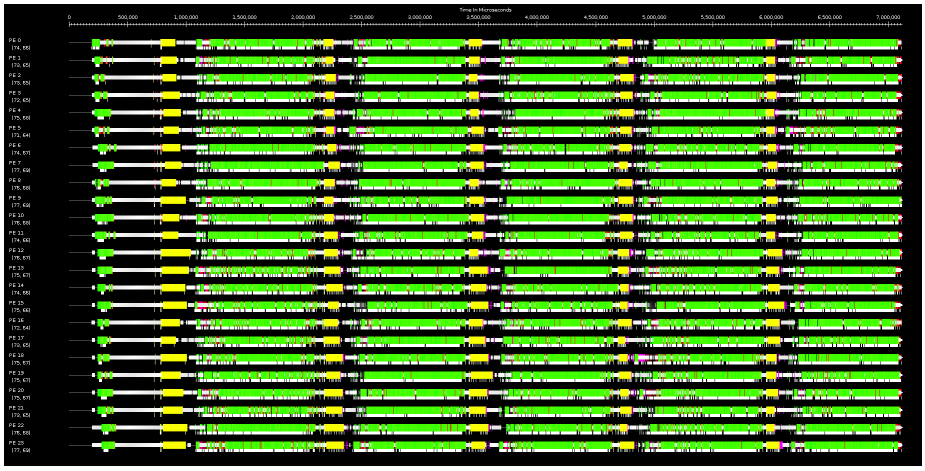

Figure 3: GreedyLB: Cores: 24, LB_PERIOD: 20, Iteration Count:100

**Are they beneficial? Why or why not? How much is the overhead?** NOT beneficial. The following table with different LB_Period values shows that GreedyCommLB's total execution times are more than those with NoLB and RefineLB. The reason for this is same as the reasons for GreedyLB that the overhead of migration is huge which is not beneficial in the scenario when the load is dynamically changing. Also similar to GreedyLB they end up decoupling the neighboring chares adding more to the communication load of CPU.

BUT there performance is better than GreedyLB because they are communication aware in the sense that if two chares are going to communicate much then they will be kept on the same PE.

| LB_Period | $t_{nolb}(ms)$ | $t_{rlb}(ms)(\%p_{rlb})$ | $t_{glb}(ms)(\%p_{glb})$ | $t_{gclb}(ms)(\%p_{gclb})$ |
|---|---|---|---|---|
| 10 | 6070 | 6683 (1.441) | 9830 (11.5) | 9817 (12.56) |
| 20 | 5955 | 5492 (.860) | 8503 (7.112) | 7117 (8.04) |
| 30 | 5876 | 5277 (.60) | 5989 (5.51) | 5914 (4.54) |

Table 2: The application is run on 24 cores with iteration Count 100. $t_{nolb}$, $t_{rlb}$, $t_{glb}$ *and* $t_{gclb}$ are the executions times of NoLB, RefineLB, GreedyLB and GreedyCommLB respectively and $p_{rlb}$, $p_{glb}$ *and* $p_{gclb}$ are the % CPU utilization for the migrations of RefineLB, GreedyLB and GreedyCommLB respectively.

Also like GreedyLB, as the LB_Period decreases, the overhead of load balancing diminishes and so the total execution time. Also with higher iteration count, the performance is bad as compared to both NoLB and RefineLB, **BUT** better that GreedyLB.