Hyper Quick Sort

Hyper quick sort is an implementation of quick sort on hypercube. Its steps are as follows −

- Divide the unsorted list among each node.
- Sort each node locally.
- From node 0, broadcast the median value.
- Split each list locally, then exchange the halves across the highest dimension.
- Repeat steps 3 and 4 in parallel until the dimension reaches 0.

Algorithm

```
procedure HYPERQUICKSORT (B, n)
begin

  id := process's label;

  for i := 1 to d do
    begin
    x := pivot;
    partition B into B1 and B2 such that B1 ≤ x < B2;
    if ith bit is 0 then

    begin
      send B2 to the process along the ith communication link;
      C := subsequence received along the ith communication link;
      B := B1 U C;
    endif

    else
      send B1 to the process along the ith communication link;
      C := subsequence received along the ith communication link;
```

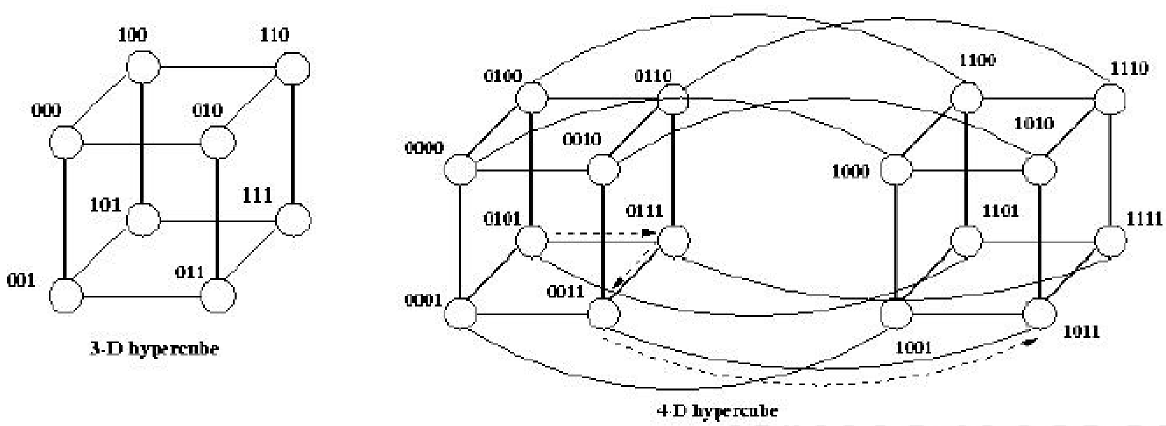
```
      B := B2 U C;
      end else
    end for

  sort B using sequential quicksort;

end HYPERQUICKSORT
```

Hypercube

Formally, *a hypercube of size n* consists of *n* processors indexed by the integers {0,1, . . . , *n* – 1}, where *n* > 0 is an integral power of 2. Processors *A* and *B* are connected *if and only if* their unique $\log_2 n$-bit strings differ in *exactly* one position.



Algorithm 1

- We randomly choose a pivot from one of the processes and broadcast it to every process.
- Each process divides its unsorted list into two lists: those smaller than (or equal) the pivot, those greater than the pivot.
- Each process in the upper half of the process list sends its “low list” to a partner process in the lower half of the process list and receives a “high list” in return.
- Now, the upper-half processes have only values greater than The pivot, and the lower-half processes have only values smaller than the pivot.
- Thereafter, the processes divide themselves into two groups and the algorithm recurses.
- After logP recursions, every process has an unsorted list of values completely disjoint from the values held by the other processes.
  - The largest value on process i will be smaller than the smallest value held by process i + 1.
  - Each process can sort its list using sequential quicksort.

Algorithm 2(My Implementation)

- Each process starts with a sequential quicksort on its local list.
- Now we have a better chance to choose a pivot that is close to the true median.
  - The process that is responsible for choosing the pivot can pick the median of its local list.
- The three next steps of hyper quick sort are the same as in parallel algorithm 1
  - Broadcast
  - Division of “low list” and high list”
  - Swap between partner processes
- The next step is different in hyper quick sort.
  - On each process, the remaining half of local list and the received half-list are merged into a sorted local list.
- Recursion within upper-half processes and lower-half processes.

Expected Case Running Time

$$\Theta\left(N \log N + \frac{d(d+1)}{2} + dN\right).$$

The *N* log *N* term represents the sequential running time from Step 2. The *d*(*d* + 1)/2 term represents the broadcast step used in Step 4. The *dN* term represents the time required for the exchanging and merging of the sets of elements.

Observations

Log P steps are needed in the recursion.

- The expected number of times a value is passed from one process to another is log P / 2 , that is quite some communication overhead!
- The median value chosen from a local segment may still be quite different from the true median of the entire list.

Although better than parallel quicksort algorithm 1, load imbalance may still arise.

Solution:

- Algorithm 3 - parallel sorting by regular sampling

Limitations

The number of processors has to a be a power of 2.

Very High communication overhead.