Extra Credit Project: Levenshtein Distance

Parallel Computing

Due on Reading day (Dec 8th), no extension will be granted.

1 Levenshtein Distance

The levenshtein distance is used to compute the distance between two strings `s1` and `s2` of length n and m. Sequentially it is computed as:

```
int levenshtein (int n, char s1[], int m, char s2[], int val[][]) {
  val [0][0] = 0;
  for (int i=1; i<=n; ++i) {
    val[0][i] = 0;
  }
  for (int i=1; i<=m; ++i) {
    val[i][0] = 0;
  }

  for (int i=1; i<=m ++i) {
    for (int j=1; j<=n ++j) {
      val[i][j] = min (val[i][j-1]+1, val[i-1][j]+1,
                       val[i-1][j-1] + (s1[j-1] != s2[i-1]));
    }
  }

  return val[m][n];
}
```

The problem is to provide parallel implementation of this algorithm and to evaluate their speedup. You can use random strings as input.

Work to accomplish:

- provide a shared-memory parallel implementation of Levenshtein.
- provide a distributed-memory parallel implmentation of Levenshtein.
- provide a GPU parallel implementation of Levenshtein.
- provide strong and weak scaling experiments at different size.

The work will be judged based on the performance achieved at different size. You can turn in only some of the implementations if you wish for partial credit.

2 Some hints

Remember how to extract parallelism from this problem. We discussed it when we discussed the midterm! The parallelism sits on the anti-diagonal of the matrix.

One can compute the Lenvenshtein distance of strings of many GBs by noticing that only `val[m][n]` needs to be returned. There is no need to keep the entire `val` array in memory.

Rather than considering each entry of the matrix as a task to execute, blocking the matrix helps having more work per task.