Olden Benchmarks Suite

MST

Benchmark

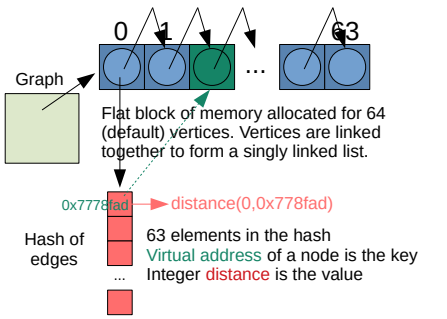
Minimum Spanning Tree (MST)

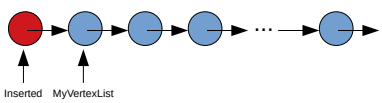
Overview

MST weight computation of a complete graph. Computation is approximate and the algorithm looks like it might finish with incorrect result. Nevertheless the benchmark can be used for the purpose of computational workload. Graph is represented as a linked list of vertices. Every node in the list has a hash table of incident edges. Graph is complete: each vertex is connected to all other vertices in the graph (except itself). Algorithm repeatedly traverses the list of vertices and gradually accumulates the MST weight. On every traversal algorithm picks the node in the list to use as an input for the next traversal. In that sense, there is a cross iteration/traversal dependency. The code below summarises the benchmark.

```
vertex = list;
list = list->next;
while (num_vertices) {
        ret = traverse_vertex_list(vertex, list);
        mst_weight += ret.distance; // accumulate the final result
        vertex = ret.vertex; // next vertex to measure the distance against
        num_vertices--;
}
return mst_weight;
```

Data Structures





BlueRule: traverses MyVertexList linearly and compares *Inserted* vertex against all vertices in the list. Finds the vertex with either minimum distance from Inserted vertex or with the minimum distance recorded so far for that vertex and returns it out of BlueRule() to be the next Inserted vertex. For all vertices algorithm records the minimum discovered distances.

```
if (distance(Inserted, vertex) < min_dist) {
        min_dist = distance(Inserted, vertex);
        ret = vertex
        vertex.min_dist = min_dist;
}
```

MST benchmark operates on a graph. All vertices are allocated together in a single chunk of memory and linked to form a singly linked list. Every vertex contains a hash of edges incident to it. Edges connect the vertex with all other vertices (except itself) in the graph. Hash uses virtual addresses of allocated vertices as keys and stores their distances from the given vertex as integer values.

Computational method

Algorithm accumulates minimum spanning tree weight value repeatedly. The number of repetitions is equal to the number of vertices in the graph. On every repetition the algorithm does a full linked list traversal and picks the vertex in the list with the minimum distance from the one given as an input to the algorithm (or with the minimum distance recorded so far for the vertex being traversed! - can cycle the algorithm on just a single vertex with the smallest in the graph incident edge).

Optimization

The algorithm is parallelizable in a divide and conquer manner. Linked list can be split into several parts. These parts can be traversed independently with the minimum element picked out of each part. The merge procedure must find the mimimum out of

all min elements and return it. Moreover, keeping the graph nodes in the linked list is absolutely not necessary – we can use a flat array instead and remove the pointer chasing code.