Implementing generalized Deep-Copy in MPI Supplementary Results

Joss Whittle*
Swansea University, UK
csjoss@swansea.ac.uk

Rita Borgo
Swansea University, UK
r.borgo@swansea.ac.uk

Mark Jones
Swansea University, UK
m.w.jones@swansea.ac.uk

1. COMPARISON WITH THE BOOST SERIALIZATION LIBRARY

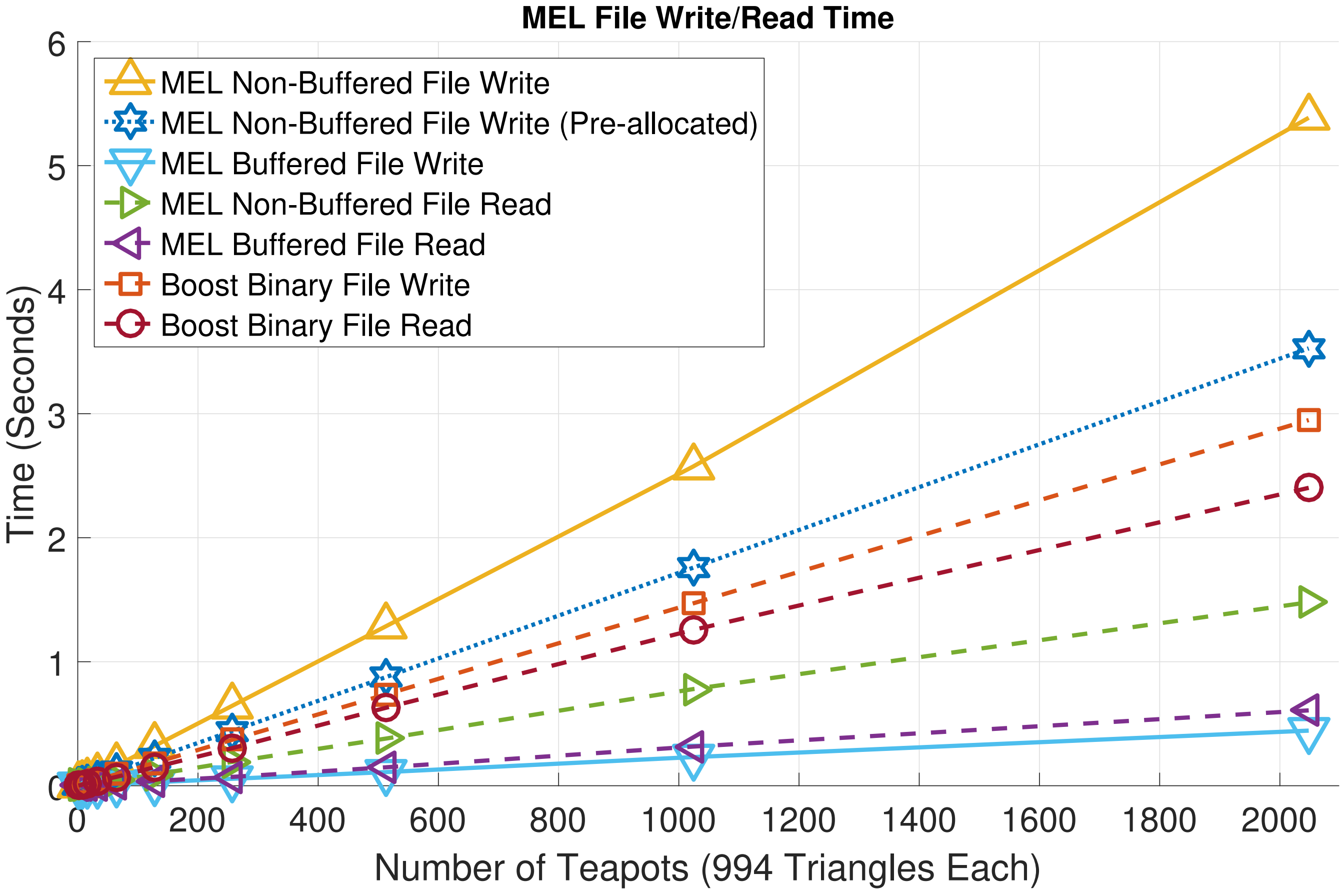


Figure 1: Time comparison of writing/reading large BVH-Tree structures to/from file.

2. PERFORMANCE OF DEEP COPYING GRAPHS POTENTIALLY CONTAINING CYCLES

2.1 Fully Connected Graph

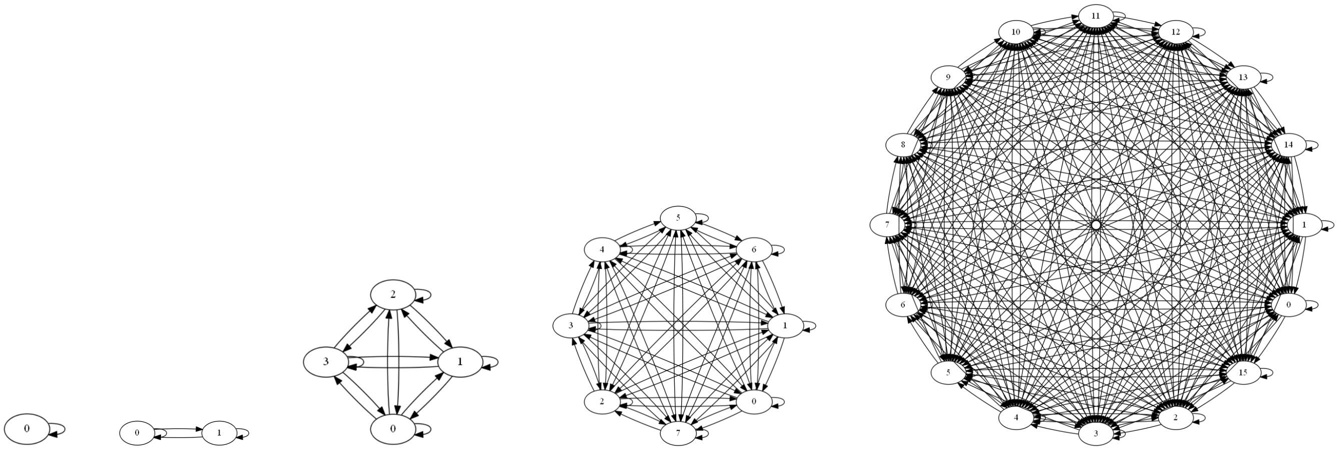
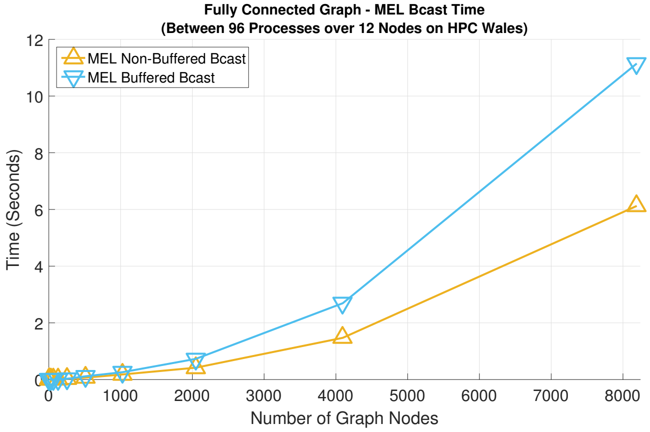
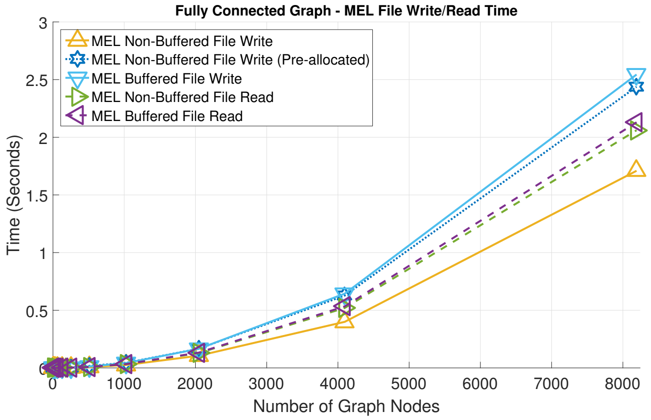


Figure 2: Fully Connected Graph for $\{2^0, 2^1, 2^2, 2^3, 2^4, ...\}$ nodes



(a) Time comparison of broadcasting large fully connected graph structures between processes within node and on separate nodes.



(b) Time comparison of writing/reading large fully connected graph structures to/from file.

2.2 Random Graph

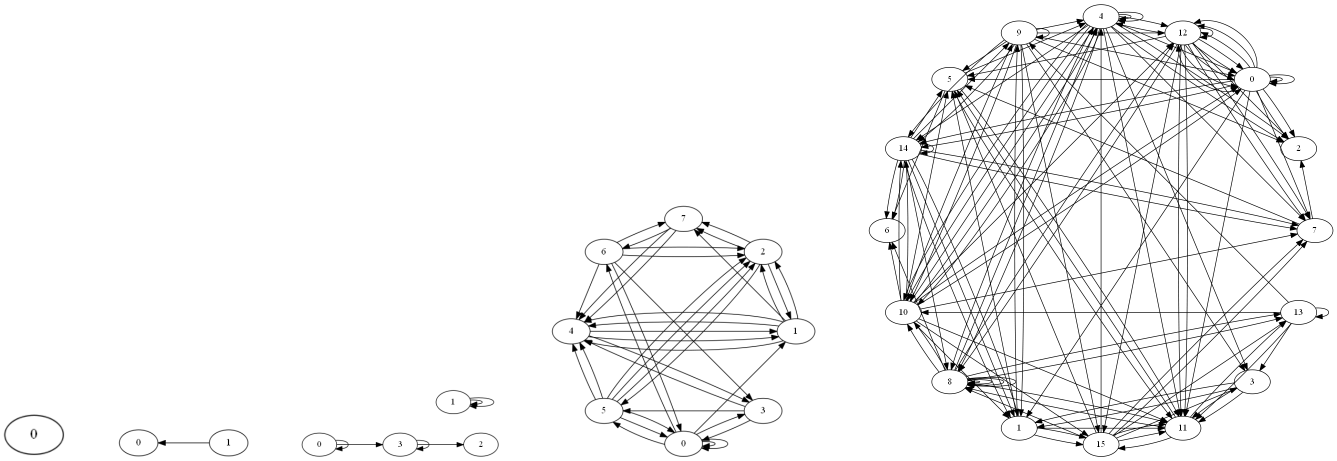
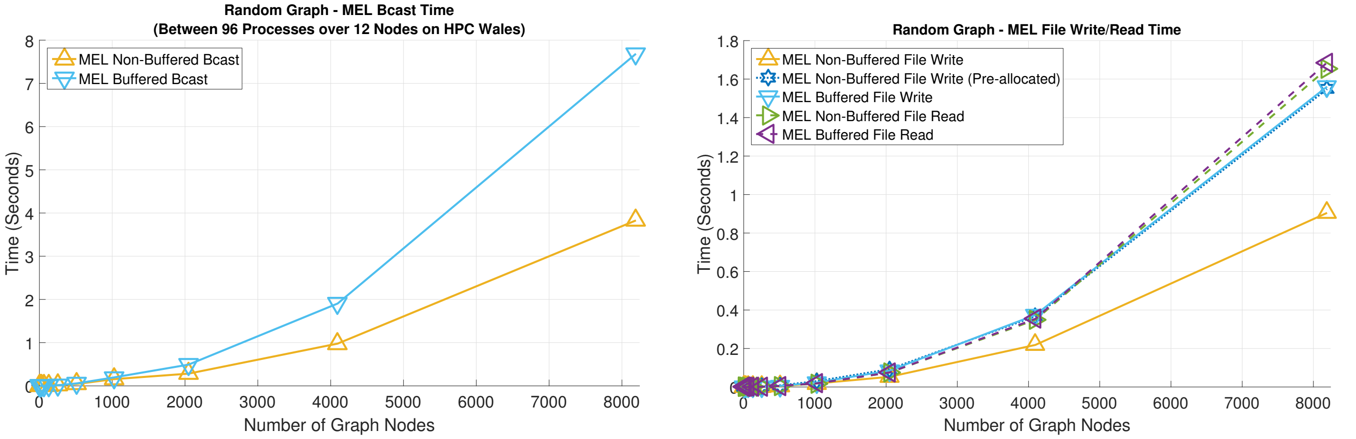


Figure 3: Random Graph for $\{2^0, 2^1, 2^2, 2^3, 2^4, ...\}$ nodes



(a) Time comparison of broadcasting large random graph structures between processes within node and on separate nodes.

(b) Time comparison of writing/reading large random graph structures to/from file.

2.3 Ring Graph

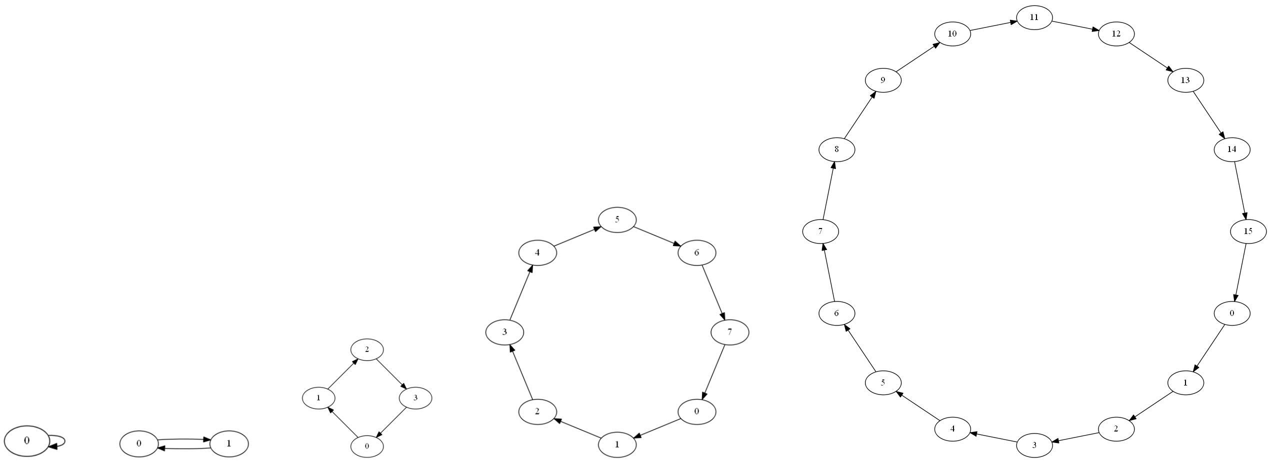
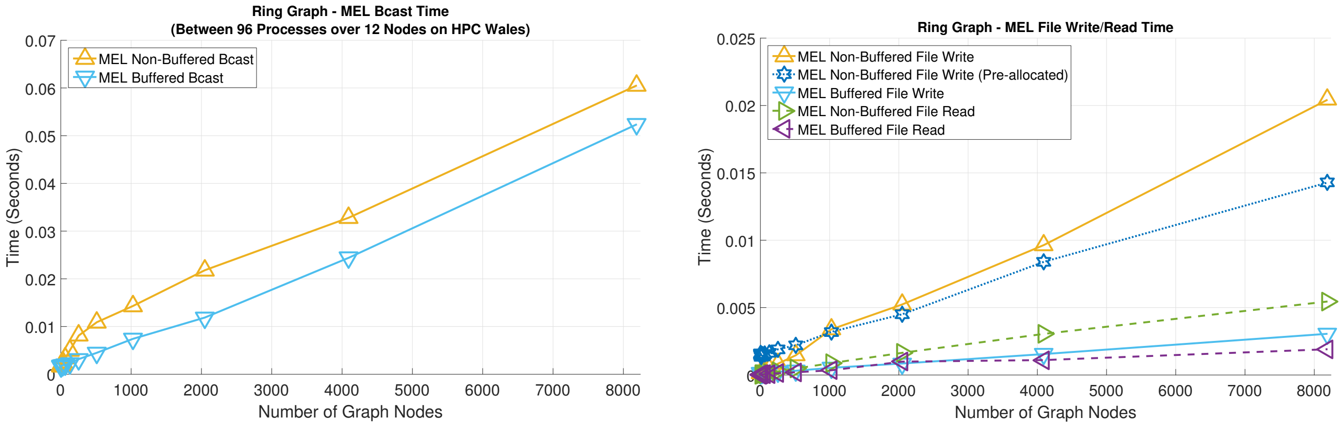


Figure 4: Ring Graph for $\{2^0, 2^1, 2^2, 2^3, 2^4, ...\}$ nodes



(a) Time comparison of broadcasting large ring structures between processes within node and on separate nodes.

(b) Time comparison of writing/reading large ring structures to/from file.

2.4 Binary Tree Graph

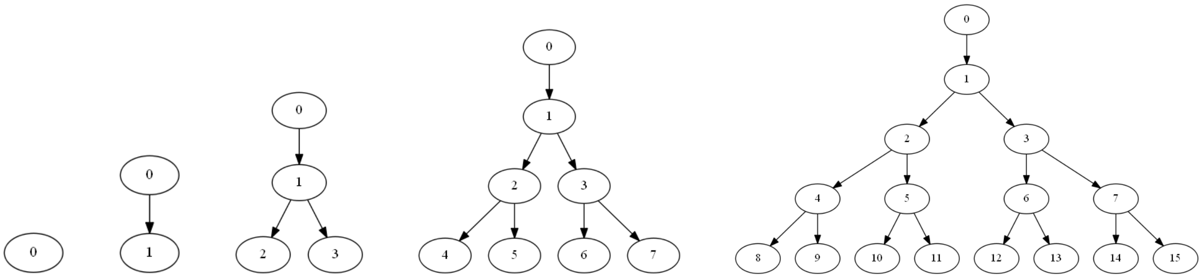
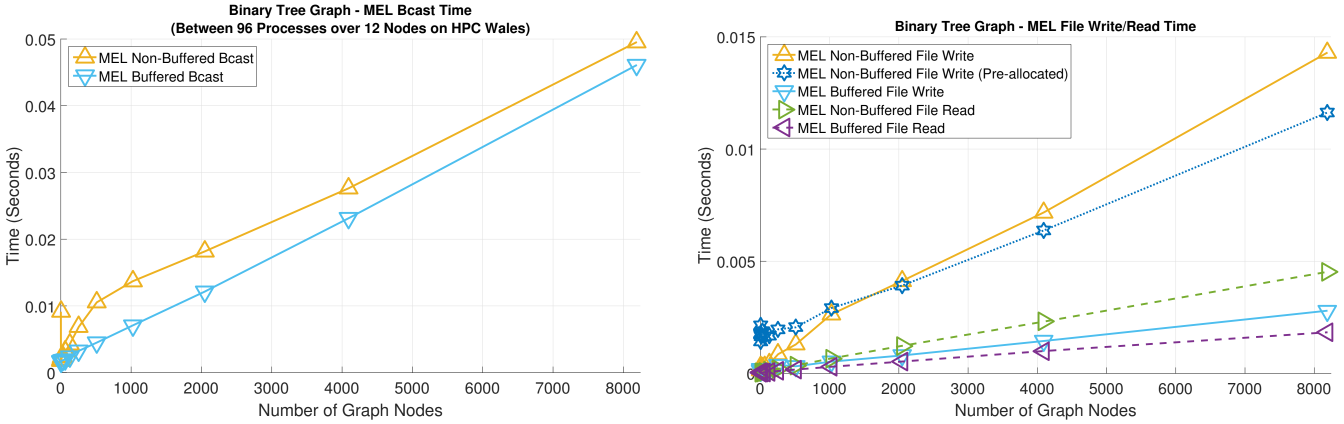


Figure 5: Binary Tree Graph for $\{2^0, 2^1, 2^2, 2^3, 2^4, ...\}$ nodes



(a) Time comparison of broadcasting large tree structures between processes within node and on separate nodes.

(b) Time comparison of writing/reading large tree structures to/from file.

APPENDIX

A. IMPLEMENTATION OF A GENERIC DIRECTED GRAPH STRUCTURE FOR DEEP COPY

```cpp
template<typename T>
struct DiGraphNode {
    /// Members
    T value;
    std::vector<DiGraphNode<T>*> edges;

    DiGraphNode() {}
    DiGraphNode(const T &_v) : value(_v) {}

    inline int getOutDegree() const {
        return (int) edges.size();
    }

    inline void addEdge(DiGraphNode<T> *node) {
        edges.push_back(node);
    }

    inline void DeepCopy(MEL::Deep::Message &msg) {
        /// Transport the vector of dangling "shared" pointers
        msg & edges;

        /// Resolve the "shared" pointers, transporting them as needed
        for (auto it = edges.begin(); it != edges.end(); ++it) {
            msg.packSharedPtr(*it);
        }
    }
};

template<typename T>
struct DiGraph {
    /// Members
    std::vector<DiGraphNode<T>*> nodes;

    DiGraph() {}
    ~DiGraph() {
        for (auto it = nodes.begin(); it != nodes.end(); ++it) {
            MEL::MemDestruct(*it);
        }
    }

    inline void addNode(const T &value) {
        nodes.push_back(MEL::MemConstruct<DiGraphNode<T>>(value));
    }

    inline DiGraphNode<T>* getNode(const int nodeId) const {
        return nodes[nodeId];
    }

    inline void DeepCopy(MEL::Deep::Message &msg) {
        /// Transport the vector of dangling "shared" pointers
        msg & nodes;

        /// Resolve the "shared" pointers, transporting them as needed
        for (auto it = nodes.begin(); it != nodes.end(); ++it) {
            msg.packSharedPtr(*it);
        }
    }
};
```

Listing 1: Deep-Copy of Generic Direct Graph