Communication-Centric Optimizations by Dynamically Detecting Collective Operations

Timo Schneider and Torsten Hoefler
University of Illinois at Urbana-Champaign

Compile

Performance of different published allreduce implementations in CAF
- The optimized version, where allreduce is detected and replaced by an MPI call is an order of magnitude faster

Rewrite with optimized Collectives

A collective operation can be described by the set of SST tuples it consists of:
- Can be used to match tuples to collectives
- Collectives have to be matched in the order of their expressiveness

Global graph is used solve the dataflow
- Dataflow is expressed in SST Tuples (cf. Single Static Assignment)
- SST is created by visiting the graph top to bottom
- Tuples can be split or merged

Most optimizations require knowledge of the global communication graph:
- Local graphs are gathered
- Dependencies stay intact as they are process local
- Send and receive operations are linked together (green arrows) in a matching step

GOAL_Compile() creates a local communication graph for each process
- At runtime, buffer addresses are available
- Note that there are no dependencies in this example

Compiler transforms this into GOAL code:
- Pattern expressed as dependency graph
- Vertices: Send / Recv operations
- Edges: Dependencies between operations
- Optimization applied in GOAL_Compile()