Distributed Graph Topology
- a necessary fix for MPI-2.2 -

Torsten Hoefler and Jesper Larsson Traeff

Open Systems Lab
Indiana University
Bloomington, USA

9^{th} MPI Forum Meeting April'09

Chicago, IL, USA

April, 6-8th 2009

Motivation - The Problem

- MPI-2.1: each process needs to specify the whole communication graph
- needs $\mathcal{O}(P^2)$ memory on each process
- even (low degree) sparse graphs still need $\Omega(P)$ memory (they tend to be connected)
- $\Rightarrow$ inherently not scalable (think of BG/L or future architectures)
- the graph interface is rarely used today because of such issues
- many applications could use it (see next slide)

Motivation - Applications

- typical application examples:
  - user-defined collective operations
  - sparse matrix computations
  - graph problems that do not change the graph structure (e.g., PageRank, Connected Components)
  - any domain decomposition with “halo zones”
- “maybe” applications:
  - graph problems that change structure (e.g., Δ Stepping)
  - adaptive mesh refinement codes

Why MPI-2.2

- we have a (known) performance (scalability) bug in MPI-2.1
- fixing it is simple
    - can be implemented on top of the old interface (see source-code)
    - does not change anything, just added functionality
    - very minor impact to implementations
    - → well within the MPI-2.2 scope
- it might have a huge impact on usability and optimizability

Why are you rushing it now?

We are not rushing it! The proposal has been in its current form (minus minor changes) waiting for feedback since Sept. 2008 (6 months). We need to meet the tight MPI-2.2 schedule which is closing down during this meeting.

Why MPI-2.2

- we have a (known) performance (scalability) bug in MPI-2.1
- fixing it is simple
  - can be implemented on top of the old interface (see source-code)
  - does not change anything, just added functionality
  - very minor impact to implementations
  - → well within the MPI-2.2 scope
- it might have a huge impact on usability and optimizability

Why are you rushing it now?

We are not rushing it! The proposal has been in its current form (minus minor changes) waiting for feedback since Sept. 2008 (6 months). We need to meet the tight MPI-2.2 schedule which is closing down during this meeting.

Two Proposals in a Nutshell

1. MPI_DIST_GRAPH_CREATE(comm_old, n, sources, degrees, destinations, weights, info, reorder,comm_graph)
    - **any** process can contribute **any** edge
    - see ticket #33 for details
    - we call it “flexible specification”

2. MPI_DIST_GRAPH_CREATE(comm_old, indegree, sources, sourceweights, outdegree, destinations, destweights, comm_graph)
    - **each** process must contribute **all** its incoming **and** outgoing edges
    - see ticket #147 for details
    - we call it “adjacent specification”

Pro/Con Option 1 (flexible specification)

1. Pro:
   - easy for users (fully subsumes old interface, i.e., if full graph is only available at a single node)
   - fully subsumes the “adjacent specification”
   - high flexibility
2. Con
   - might look complex (we doubt that since it’s effectively the same as the MPI-2.1 specification)
   - causes communication overhead during creation

Pro/Con Option 2 (adjacent specification)

1. Pro:
   - might look simpler (again, we doubt that)
   - avoids communication during construction

2. Con
   - significantly less flexible and less convenient
   - forces the user to distribute all information about adjacent processes (they usually know outgoing edges - but incoming?)
   - doubles the size of the graph on the user side

And why not both?

1. Pro:
   - sum of all pros from prev. slides
2. Con
   - cluttered interface (but ok, we do have the *v collectives ;)
   - more options → steeper learning curve
   - more maintenance (testing etc.) overhead

Big Question: can we live with the flexible specification only (since it subsumes the adjacent)?
Answer: only if the Cons can be addressed :)

Addressing the Cons! (part 1)

1. might look too complex - let's discuss it!
   - *current:* MPI_GRAPH_CREATE(comm_old, nnodes, index, edges, reorder, comm_graph)
   - *flexible:* MPI_DIST_GRAPH_CREATE(comm_old, n, sources, degrees, destinations, weights, info, reorder,comm_graph)
   - only one input array (sources) more than current interface!
   - current interface is very limited because it misses the sources array! (try to specify a "hole" in the middle of a comm)
   - *adjacent:* MPI_DIST_GRAPH_CREATE(comm_old, indegree, sources, sourceweights, outdegree, destinations, destweights, comm_graph)
   - adjacent is fundamentally different from current interface

Addressing the Cons! (part 2)

1. needs to communicate - that’s true, but:
   - communicator creation needs allreduce to figure out CID anyway (does anyone do pre-allocation?)
   - check if something needs to be communicated can be piggy-backed (e.g., global OR if somebody has any edges that don’t either originate or terminate at himself)
   - optimization requires communication anyway (what is the point of a graph comm without any optimization?)
   - we could also define an info-value that states that no communication is necessary: all_adjacent” or such
   - we don’t discuss the case where something needs to be communicated because it is clear that clever algorithms can be designed inside MPI to do this (or we can push it to the user who will most likely do an alltoall)

Looking at it in detail!

1. communication problems exist - but they’re rather small!
2. can be easily solved with a predefined info object!
3. we propose “all_adjacent” as info object that indicates that each process has complete local information
4. info objects have to be specified collectively!

Reorder - global or local?

1. allowing reorder locally (on some procs) improves flexibility tremendously
2. e.g., special I/O or co-processing needs of specific ranks
3. → all would have to say reorder = false
4. providing it locally causes communication overhead (figuring out the subgraph etc.)!
5. we should discuss an Info object, such as "all_reorder"