ABSTRACTS FOR EXPRESSING NETWORK PROBLEMS IN PETSc

Shrirang Abhyankar, Jed Brown, Matthew Knepley, Florian Meier, Barry Smith

SIAM Workshop on Network Science 2014
July 6-7 · Chicago

Summary

Developing scalable software for large-scale applications, particularly for networks and circuits, is challenging due to the underlying unstructured and irregular geometry of the problem. We present a programming framework recently added to the PETSc [2] library to easily express network problems, and thereby reduce the application development time. A brief overview of the framework is presented and two application examples, one from power grid and the other from radio networks, are discussed.

PETSc

PETSc is an open source package for the numerical solution of large-scale applications and provides the building blocks for the implementation of large-scale application codes on parallel (and serial) computers. The wide range of sequential and parallel linear and nonlinear solvers, time-stepping methods, preconditioners, reordering strategies, flexible runtime options, ease of code implementation, debugging options, and a comprehensive source code profiler have made PETSc an attractive experimentation platform for developing scientific applications. Along with the numerical solvers, PETSc also provides abstractions through the `DM` class for managing the application geometry and data.

DMNetwork framework

`DMNetwork` is a subclass of the `DM` class that provides a framework for managing geometry and data for unstructured grids, particularly suited for network applications. Its built on top of the `DMPlex` subclass, a rewritten version of the Sieve [3] framework. Delving on three basic elements of any network: nodes, edges, and data, the framework provides abstractions for easily creating the network layout, partitioning, data movement, and utility routines for extracting connectivity information. A key feature of this framework is that the user only needs to work with higher level application specific abstractions while PETSc takes care of the underlying data management; a feature consistent with the PETSc philosophy. We present the salient features of the DMNetwork framework next and list some of the utility routines in Table 1.

- Support for assigning different degrees of freedom for any node or edge. This is particularly important for networks that comprise of sub-networks having different characteristics.
- Any data, 'component' as we term it, can be attached with a network node or edge. For example, the component could be edge weights or vertex weights for graph problems, or network elements in the case of circuits. Multiple components can be attached to an edge or node.
- Support for partitioning (edge distribution) of the network graph using ParMetis or Chaco partitioners. Components associated with nodes/edges are also distributed to the appropriate processor when the network is partitioned.
- The framework can create the linear operator or the Jacobian for the network.
- Global (parallel) vectors and local vectors for residual evaluation can be created by `DMNetwork`.
- `DMNetwork` also keeps track of the global and local offsets for use in function evaluation or matrix assembly. It also stores information of the 'ghost' nodes (nodes that need to perform communication with other processors).
- Support for global-to-local and local-to-global communication of vector entries.
- While doing a calculation, most network applications require information about the edges connected to a node, and/or the nodes covering an edge. The framework provides API routines to extract this information.
- Full compatibility with all PETSc's linear (`KSP`), nonlinear (`SNES`), and time-stepping (`TS`) solvers.

Utility routine	Description
CreateLayout	Creates network graph
AddComponent	Adds a component to a node or edge
AddNumVariables	Adds the degrees of freedom for a node or edge
SetUp	Creates Network
GetNumComponents	Gets the number of components at a node or edge
Distribute	Partitions network, distributes component data
GetSupportingEdges	Gets the edges supporting a node
GetConnectedNodes	Gets nodes covering an edge

Table 1: DMNetwork API routines

Application examples

The `DMNetwork` interface has been used for a couple of nonlinear applications that also use PETSc's nonlinear solver `SNES`.

Power systems

Power flow analysis, sometimes referred to as load flow analysis, is the linchpin of steady-state power systems analysis. Power flow entails the solution of nonlinear power balance equations at each bus (node), i.e., the summation of the power injected at each bus and absorbed by the network must equate to zero. The equations for each bus are given by

$$\sum_{k \in I(i,k) \neq 0} |V_i||V_k|(G_{ik}cos(\theta_{ik}) + B_{ik}sin(\theta_{ik})) - P_i^{inj} = 0 \tag{1}$$

$$\sum_{k \in I(i,k) \neq 0} |V_i||V_k|(G_{ik}sin(\theta_{ik}) - B_{ik}cos(\theta_{ik})) - Q_i^{inj} = 0, \tag{2}$$

where $\theta_{ik} = \theta_i - \theta_k$ and $V \in \mathbb{R}^n$ and $\theta \in \mathbb{R}^n$ are the variables to be solved. In this example application, each node has two degrees of freedom, V_i, θ_i for each bus. To compute the residual at each bus, information about its neighboring buses and the connected edges is needed. This can be easily obtained using the `DMNetwork` API routines.

Figure 1 shows the scalability of the power flow application with both direct (SuperLU_Dist package) and iterative linear solvers (GMRES + restricted additive Schwarz preconditioner) for a very large 90000 bus power system.

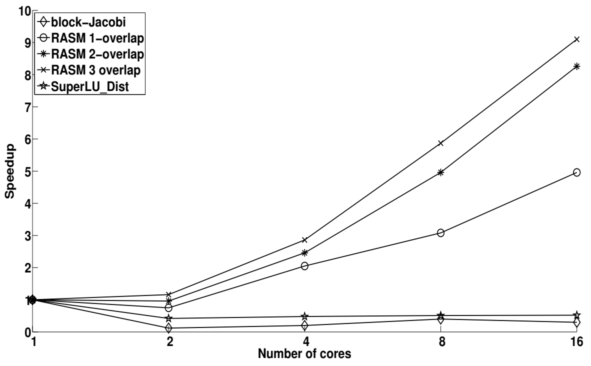

Figure 1: Speedup for power flow for a 91984 bus case with different preconditioning schemes [1]

Radio networks

The second application that currently uses the `DMNetwork` framework is a nonlinear problem from radio networks. Here, the goal is to find the probabilities of receiving packets for radio links, which are modeled as nodes in the network. Each radio link i in the network generates packets conforming to a Poisson distribution with probability g_i. These packets are routed along fixed paths to a sink. When two packets are sent at the same time, they are dropped. This builds up the following equation system:

$$f_{a,i}(x) = 0 = g_i + \sum_{j \in F_i} x_{s,j} - x_{a,i}, \tag{3}$$

$$f_{s,i}(x) = 0 = x_{a,i} \left(\prod_{j \in I_i} (1 - x_{a,j}) \right) - x_{s,i}. \tag{4}$$

where $x_{a,i}$ and $x_{s,i}$ are the probabilities that link i accesses the channel and succeeds with the transmission respectively and F_i and I_i are the inflowing and interfering links.

References

[1] S. Abyankar, B. F. Smith, and E. Constantinescu. Evaluation of overlapping restricted additive Schwarz preconditioning for parallel solution of very large power flow problems. In *3rd International Workshop on High Performance Computing, Networking, and Analytics for the Power Grid*. ACM, 2013.

[2] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2013.

[3] M. G. Knepley and D. A. Karpeev. Mesh algorithms for PDE with Sieve I: Mesh distribution. Technical Report ANL/MCS-P1455-0907, Argonne National Laboratory, February 2007. `ftp://info.mcs.anl.gov/pub/tech_reports/reports/P1455.pdf`.