PSMG: A Parallel Problem Generator for a Structure Conveying Modelling Language for Mathematical Programming

Feng Qiang[1] F.Qiang@sms.ed.ac.uk, Andreas Grothey[1] A.Grothey@ed.ac.uk

[1]University of Edinburgh, Edinburgh, United Kingdom

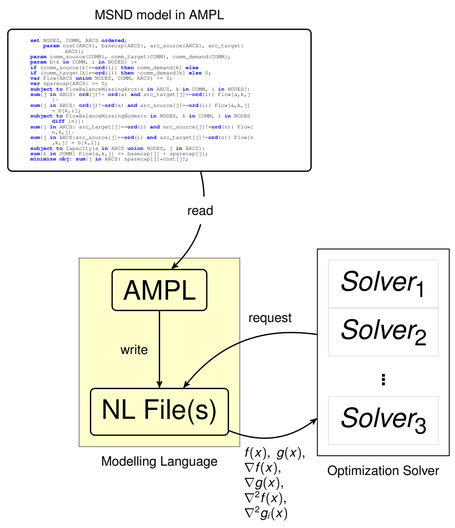
Introduction

- **The Problem:**
  - Large scale optimization problems can have millions of constraints and variables and structured.

    Structure: means that a "discernible pattern" exists in the constraint matrix. This pattern is usually the result of an underlying problem generation process, such as discretizations in space, time or probability space; many real world complex optimization problems are composed of multiple similar sub-problems and relations among the sub-problems.
  - Structure is used to facilitate parallel solving the problem faster.
  - Problem generation and evaluating of problem functions is a time and memory bottleneck.
  - Especially when the optimization solvers are running in parallel, but the model generator is not.
- **Our Solution:**
  We implemented a Parallel Structure Model Generator (PSMG) that can perform problem generation in parallel.

PSMG Overview:
- Take advantage of today's parallel computing environment.
- A generic interface for linking up with any structured exploiting solvers.
- The only parallel algebraic modelling language (as far as we are aware).
- Obtained good parallel efficiency on up to 256 processes.
- Implemented in C/C++ with MPI.

Traditional Modelling Language Approach



- Commonly, optimization problems are modelled with a **modelling language**, such as AMPL, GAMS, etc to achieve maintainability and separation of model and data.
- Modelling language provides function and derivative values for the solver.
- There are parallel solvers such as OOPS, PIPS, etc.
- ⇒ Serial model generation becomes a bottleneck.
- The need for parallel model generation has also been recognised by the European Exascale Software Initiative[2].

Structured Problem

Example: Multi-Commodity Survivable Network Flow Design (**MSND**)
In this problem the objective is to install additional capacity on the links of a transportation network so that several commodities can be routed simultaneously without exceeding link capacities even when one of the links or nodes should fail.

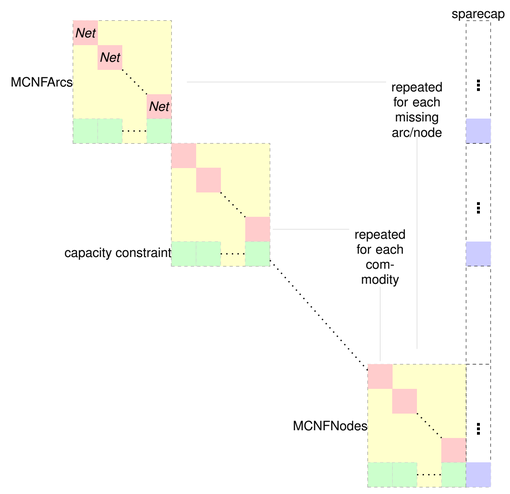


Figure:Visualisation of the structure in MSND constraint matrix

- MSND problem contains nested structures repeated for each commodity, arc, and node.
- Red blocks correspond to a *Net* block in the model file.
- Yellow blocks correspond a *MCNFArcs* or *MCNFNodes* block.
- The green and blue blocks are corresponding to the linking constraints–*the capacity constraints*.

SML Syntax

- SML syntax is based on AMPL with additional keywords such as `block` and etc[1].
- The highlighted blocks demonstrate the use of nested block statements to describe problem structures.

Model MSND in SML

```
1  set NODES, ARCS, COMM;
2  param cost{ARCS}, basecap{ARCS}, arc_source{ARCS}, arc_target{ARCS};
3  param comm_source{COMM}, comm_target{COMM}, comm_demand{COMM};
4  param b{k in COMM, i in NODES} := if(comm_source[k]==i) then comm_demand[
     k] else if(comm_target[k]==i) then -comm_demand[k] else 0;
5  var sparecap{ARCS}>=0;
6  block MCNFArcs{a in ARCS}: {
7    set ARCSDIFF := ARCS diff {a};
8    block Net{k in COMM}: {
9      var Flow{ARCSDIFF}>=0;
10     subject to FlowBalance{i in NODES}:
11       sum{j in ARCSDIFF:arc_target[j]==ord(i)} Flow[j] - sum{j in
           ARCSDIFF:arc_source[j]==ord(i)} Flow[j] = b[k,i];
12   }
13   subject to Capacity{j in ARCSDIFF}:
14     sum{k in COMM} Net[k].Flow[j] <= basecap[j] + sparecap[j];
15 }
16 block MCNFNodes{n in NODES}: {
17   set NODESDIFF := NODES diff {n};
18   set ARCSDIFF := {m in ARCS:arc_source[m]!=ord(n)
19    and arc_target[m]!=ord(n)};
20   block Net{k in COMM}: {
21     var Flow{ARCS} >= 0;
22     subject to FlowBlance{i in NODESDIFF}:
23       sum{j in ARCSDIFF:arc_target[j]==ord(i)} Flow[j] - sum{j in
           ARCSDIFF:arc_source[j]==ord(i)} Flow[j] = b[k,i];
24   }
25   subject to Capacity{j in ARCSDIFF}:
26    sum{k in COMM} Net[k].Flow[j] <= basecap[j] + sparecap[j];
27 }
28 minimize costToInstall: sum{x in ARCS} sparecap[x]*cost[x];
```

MSND Problem modelled in SML.

- Model can express problem structure corresponding to the MSND constraint matrix figure.
- Structure of problem can be easily parsed.

PSMG Design

- Prototype model tree
  - Internal representation of the nested bock dependencies defined in the model file.
  - Each node corresponds to one block in the model.
  - Each node associates with an indexing expression (to be expanded for creating expanded model tree).
  - Each node contains a list of entities declared in this block.

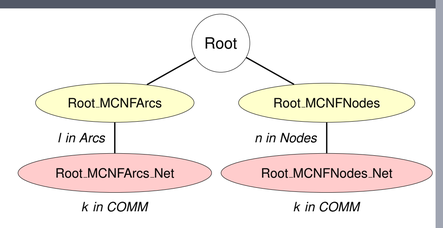


Figure: The prototype model tree for MSND problem.

- Expanded model tree
  - Obtained by creating copies of each node in the prototype tree according to the associated indexing expression and data.
  - Generated from Prototype tree after reading the problem data.
  - Represents an instance of the problem
  - Stores the structure information of the problem instance;
  - Works as interface for passing problem structure to solver(s).

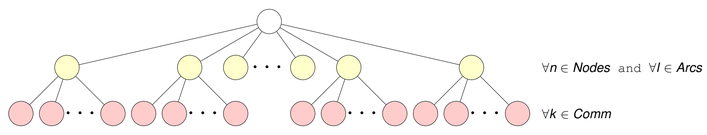


Figure:The expanded model tree for the MSND problem.

- Model context tree
  - Has the same tree structure as the expanded model tree.
  - Each node stores data of the corresponding node at the expanded model tree.
  - Implements "lazy" data initialization/computation and hierarchical data lookup features in order to optimize memory usage.

Solver Driven Work Assignment

Design Goal:
- Pass problem structure to structure exploiting solver.
- Delay as much as possible work for parallel process later.

Reason for solver to decide the work assignment:
- Since solver understands the complexity of each subproblem better.
- Solver is responsible for assigning the computation work among all the PSMG processes.
- Each PSMG processes will be able to compute different part of the problem upon the solver processes requests.

PSMG interface with the solver

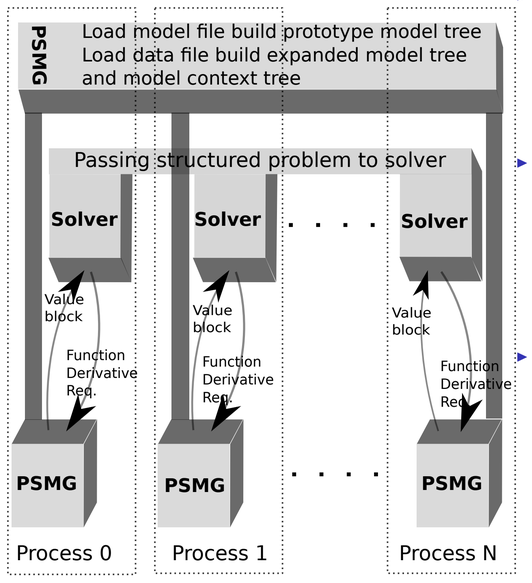


Figure: The PSMG workflow with a parallel optimization solver.

Workflow:
- Step 1: Same work on all processes:
  - Parses model and data file.
  - Build prototype tree, expanded model tree, and model context tree.
- Step 2: Passing expanded model tree to solver:
  - The structure information is passed by the expanded model tree.
  - Solver does the work assignment.
- Step 3: Function and derivative evaluation:
  - Each PSMG process handles a sub-block of the function and derivative evaluation requests by calback routines.
  - Data in model context tree is populated upon necessary – *"lazy approach"*.

Note that Step 1 and 2 are very small percentage work of the total problem generation(*less than 0.1%*).

PSMG Parallel Efficiency

- Generating a MSND problem that has 55 commodities on a network of complete graph of 30 vertex and 435 arcs (*msnd30_55*).
- The problem has 1,130,795 variables and 967,410 constraints.

Number of parallel PSMG processes	Structure Setup	Finishing Time(s)	Speedup	Efficiency
1	1.39	1911.17	NA	NA
2	1.49	1009.79	1.89	0.95
4	1.47	509.71	3.75	0.94
8	1.48	254.39	7.51	0.94
16	1.44	125.78	15.19	0.95
32	1.54	71.06	26.9	0.84
64	1.47	37.61	50.82	0.79
128	1.66	20.53	93.09	0.73
256	1.65	10.37	184.3	0.72

Table: PSMG speedup and parallel efficiency for problem `msnd30_55`

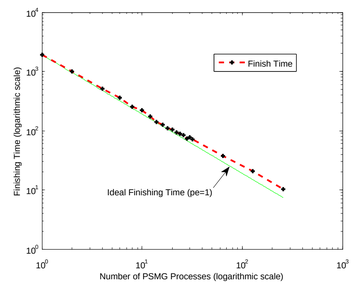


Figure: Loglog plot for parallel finishing time of PSMG for problem *msnd30_55*.

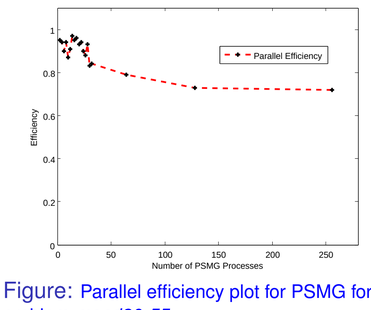


Figure: Parallel efficiency plot for PSMG for problem *msnd30_55*.

- PSMG achieved excellent speed-up on 16 processes.
- Also respectable speed-up on 256 processes.

PSMG Memory Usage Analysis

- AMPL failed to generate *msnd30_55* problem on a node with 4GB memory (because of out of memory error).
- PSMG removes single node memory limitation.
  PSMG is able to distribute the problem data on multiple nodes.

Number of parallel PSMG processes	Prototype Tree per Process (MBytes)	Expanded Tree per Process (MBytes)	Context Tree per Process (GBytes)	Memory per Process (GBytes)	Total Memory (GBytes)	Structure Memory Overhead
1	0.05	9.54	4.20	4.21	4.21	0.22%
2	0.05	9.54	2.10	2.11	4.22	0.44%
4	0.05	9.54	1.05	1.06	4.24	0.88%
8	0.05	9.54	0.53	0.54	4.28	1.75%
16	0.05	9.54	0.26	0.27	4.36	3.44%
32	0.05	9.54	0.13	0.14	4.52	6.63%
64	0.05	9.54	0.07	0.08	4.83	12.41%
128	0.05	9.54	0.03	0.04	5.45	21.97%
256	0.05	9.54	0.02	0.03	6.70	35.75%

Table: Memory usage information for generating a *msnd30_55* problem

- Memory Overhead contains:
  - Data in prototype and expanded tree which is repeated on every processor.
  - Structure overhead for context tree.
- Note that data in context tree is distributed.

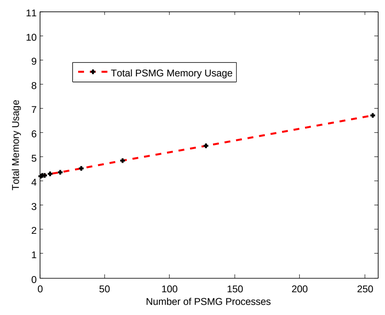


Figure: Total memory usage plot for generating problem *msnd30_55*.

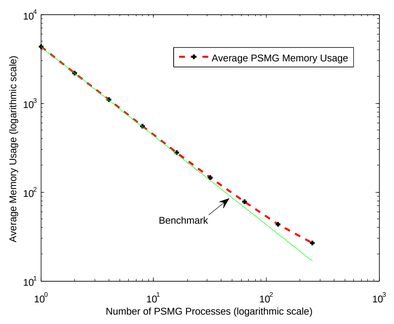


Figure: Per processor memory usage for generating problem *msnd30_55*.

Conclusion and Future Work

Advantages of PSMG:
- Easy-to-use syntax to model optimization problems that are composed of nested sub-problems.
- Passing the problem structure information to the optimization solver.
- Parallelise the problem generation process using the structure defined in the model.
- Solver driven parallel interface enables the parallel solver to achieve load balancing and data locality.

SML/PSMG is only parallel AML as far as we are aware.

- As this time, PSMG only supports parallel generation of linear problems, the work to make PSMG support nonlinear programming is currently on-going.

Reference

[1] Colombo, M., Grothey, A., Hogg, J., Woodsend, K., Gondzio, J.: A structure-conveying modelling language for mathematical and stochastic programming. Mathematical Programming Computation 1 (2009) 223247

[2] European Exascale Software Initiative: Final report on roadmap and recommendations development. http://www.eesi-project.eu (2011)