Parallelization Issues for MINLP

Bill Hart
Cindy Phillips
Sandia National Laboratories
{wehart,caphill}@sandia.gov

IMA Workshop, November 2008
Acknowledgements

P. K. Polisetty
E. P. Gatzke
E. Boman
L. Jiang
D. M. Gay
J. Eckstein
Overview

- Parallel hardware
- Parallelization issues
- A case study: GNLP
- PEBBL framework
  - Why you shouldn’t roll your own parallel branch-and-bound framework...
Parallel Computing Platforms...
Multicore

A multicore processor includes multiple execution units ("cores"), which can issue multiple instructions per cycle from multiple instruction streams.

- Shared interface to bus and L2 caches
- Shared-memory programming model
Symmetric Multiprocessors (SMP)

A SMP is a computer system with multiple identical processors that share memory and connect via a bus.

- Limited number of CPUs
- Shared-memory programming model

IBM Multi-chip Module (MCM) 8 CPUs
Cluster Computing

A cluster is a group of loosely coupled computers that work together closely, but which are managed as a single computer.

- Message-passing programming model
- Small clusters often have both local and shared disk space
- Hybrids are increasingly common (e.g. SMP clusters)
Massively Parallel Computing

A massively parallel (MP) computer is a tightly-coupled computer with many processors communicating on a high-performance network.

- Message-passing programming model
- No local or shared disk
- Dedicated compute nodes for I/O
- Hybrids are increasingly common (e.g. SMP clusters)

ASCI Red Storm
Grid Computing

Grid computing dynamically couples computers communicating over the Internet to work on a given problem.

- Can use message passing, but reliability can be a problem
- High latency, low bandwidth communication
- Available compute resources may change dynamically
- Simple master-worker parallelism is common
Cloud computing allows users to access computing services from the internet without detailed knowledge (or control) of the computing process.

- Parallel computing can be provided as a service
- Nature of parallel computing may depend on the CC vendor
  - E.g. Amazon EC2 can look like a compute cluster
- May be able to dynamically reserve new computing resources
What Can We Parallelize in MINLP Solvers?
Independent Subcomputations

Independent execution of...

- LP solves
- NLP solves
- Generating envelopes & envelope refinement
- Cut generation
- Branch evaluation (pseudo-cost initialization or strong branching)
- Constraint propagation
- MILP solves
Parallel Subcomputations

Parallel execution of...

- Linear algebra
- LP solve
- MILP solves
- NLP solves
Hybrid/Heterogeneous Parallelization Schemes

Distributed Data Management
- Use a distributed store to leverage distributed memory
- Goal: work with a larger set of active subproblems

Decentralized algorithmic control
- A fundamentally different software architecture
- May raise new algorithmic challenges (e.g. termination)

Parallel decomposition

Simultaneous management of independent branching trees...
What Might Parallel MINLP Solvers Do Differently?

Aggressive branch evaluation
- Idea: perform strong branching in parallel
- Overhead of additional work can be mitigated through parallel computation

Aggressive incumbent generation
- Can use “free” cycles during ramp up
- Can hide communication latencies
A Case Study: GNLP
Application Example

Goal: design hardware to prevent explosives accidents in extreme conditions

Problem Formulation:
- Bound maximal probability of electronics hardware malfunctions
- Rigorous global optimization is needed to certify that the level of risk is acceptable.

\[
\begin{align*}
\max & \quad x_1^2 x_5^4 (x_1^4 + 3x_1^2 x_2^2 + 2x_1 x_3 + x_2^2 + x_4)(x_1^2 + x_2)^3 (x_5^4 + 3x_5^2 x_6 + 2x_5 x_7 + x_6^2 + x_8)^2 \\
\sum_{i=1}^{8} x_i &= 1 \\
x_i &\geq 0
\end{align*}
\]
Parallel Global NLP (GNLP)

Idea: Leverage Sandia’s PEBBL library to develop a parallel branch-and-bound engine to globally optimize nonconvex NLP and MINLP problems.

Strategy:
- Analysis of nonconvex bounds with
  - Reformulations
  - Linear-programming relaxations
- Leverage COCONUT software for expression-tree analysis
- Leverage COIN-OR for LP solves

Note: similar strategy used by BARON and Coenne
COCONUT

- European research project implemented in C++
- Utilizes FILIB and MTL
- Optimization problems are represented using Directed Acyclic Graphs (DAG)
- Supports a wide variety of techniques:
  - Variable range reduction using constraint propagation (Interval Arithmetic)
  - Automatic differentiation
  - Generates linear relaxations using interval slopes
  - Exclusion boxes to eliminate infeasible portions of the search space

Idea: leverage COCONUT as a library for expression tree analysis.
LP-Based Relaxations

Perform McCormick / Smith and Pantelides convexification of all nonlinear constraints

- New variables and constraints are simple nonlinear expressions
- Bilinear and fractional function relaxations already linear
- Other nonlinear functions are monotonic functions
  • Linearize at upper and lower bounds + additional points

\[ y = -x^2 \]

After partitioning, linearizations are updated based on new bounds

(Tawarmalani and Sahinidis, 2000)
Parallel Branch-and-Bound

The PEBBL branch-and-bound library provides an infrastructure for scalable parallel branch-and-bound.

- PEBBL development was initially developed for the PICO MILP solver.
- PEBBL was extracted to support projects like GNLP.

Specific applications

GNLP – Global Nonlinear Programming solver

PEBBL -- *Parallel Enumeration and Branch and Bound Library*
Generic parallel branch and bound
Scalable Parallelism

Key Ideas:

- Coordinate search with multiple hubs
- Hubs interact with workers, who do work and store subproblems
- Hubs interact to balance subproblem quality
Extreme Case: Centralized control
Extreme Case: Decentralized Control
PEBBL for Different Compute Platforms...

- **Multicore/SMP**
  - May be able to apply MPI parallelism across the cores/processors
- **Clusters**
  - Balance hub/worker ratio can achieve large-scale scalability
- **MPP**
  - Can achieve greater scalability, probably with more hubs
- **Grid**
  - Not a good match (because of dependence on MPI)
  - On a reliable “local grid”, PEBBL works well with coarse-grain workloads
- **Cloud**
  - PEBBL can be tuned for the size of the compute resources that have been reserved
GNLP Implementation

Idea: derive GNLP solver from PEBBL classes
- Core application: bounding, branching, etc.
- Parallel application: define un/packing utilities
GNLP Status

Prototype solver “finished” fall, 2007
- Ran out of funding (small project)
- Ran out of patience with COCONUT

Capabilities
- PEBBL extension
- Generation of convex envelopes
- Integration of NLP incumbent generators
- Range reduction
- *Mixed-integer not supported*

Application results
- Maximum likelihood application(s) solved on compute clusters
- Parallel scalability seemed fine
Parallelization with PEBBL
The PEBBL and PICO Libraries

- Specific applications

- PICO -- Parallel Integer and Combinatorial Optimization
  Specific to mixed integer programming

- PEBBL -- Parallel Enumeration and Branch and Bound Library
  Generic parallel branch and bound

- Jonathan Eckstein (Rutgers) PEBBL core designer/developer
PEBBL Features for Efficient Parallel B&B

- Efficient processor use during ramp-up (beginning)
- Integration of heuristics to generate good solutions early
- Worker/hub hierarchy
- Efficient work storage/distribution
- Load balancing
- Non-preemptive proportional-share “thread” scheduler
- Correct termination
- Early output
- Checkpointing
What To Do With 9000 Processors and One Subproblem?

Option 1: Presplitting

Make log P branching choices and expand all ways (P problems)  

P = # processors

BAD!

Expands many problems that would be fathomed in a serial solution.
PICO MIP Ramp-up

- Serialize tree growth
  - All processors work in parallel on a single node
- Parallelize
  - Preprocessing
  - LP bounding
  - Cutting plane generation
  - Incumbent Heuristics
    - Eckstein/Nediak
    - feasibility pump
    - LP-based approximation algorithms
    - Fractional decomposition tree
  - Splitting decisions
    - Pseudocost (strong-branching-based) initialization
- Offline root LP solve
Crossing over

- Switch from parallel operations on one node to processing independent subproblems (serially)

- Work division by processor ID/rank
- Crossover to parallel with perfect load balance
  - When there are enough subproblems to keep the processors busy
  - When single subproblems cannot effectively use parallelism
Hubs and Workers

- Control communication
  - Processor utilization
  - Approximation of serial order

- Subproblem pools at both the hubs and workers
- Heap (best first), stack (depth first), queue (breadth first) or custom
- Hubs keep only *tokens*
  - Subproblem identifier
  - Bound
  - Location (processor, address)
Subproblem Movement: Hub-controlled

Hub ➔ Worker
- When worker has low load or low-quality local pool

Worker ➔ Hub
- Hub draws work back when it is out of work and the cluster is unbalanced
Subproblem Movement: Worker-Controlled

- Send new subproblem tokens to hub (probabilistically) depending on load
- Probabilistically scatter tokens to a random hub. If load in cluster is high relative to others, scatter probability increases.
Subproblem Movement/Data Storage
Load Balancing

- Hub pullback
- Random scattering
- Rendezvous
  - Hubs determine load (function of quantity and quality)
  - Use binary tree of hubs
  - Determine what processors need more work or better work
  - Exchange work
Reduction: Tree-based Implementation

- Local load (quantity here, but usually quality too)
- Tree on hubs

This hub’s load

Subtree total load

\[ \text{average load} = \frac{40}{8} = 5 \]
Broadcast: All processors Learn Average Load

\[
\text{average load} = \frac{40}{8} = 5
\]
Parallel Prefix (Prefix-sum, Scan)

- Introduced by Blelloch [1990].
- “Sum” is binary associative (+, *, min, max, left-copy)
- Applications: lexically comparing strings of characters, adding multiprecision numbers, evaluating polynomials, sorting and searching.

\[
\begin{array}{c|c|c|c|c}
P0 & A & B & C & D \\
P1 &   & B &   &   \\
P2 &   &   & C &   \\
P3 &   &   &   & D \\
\end{array}
\]

Scan

\[
\begin{array}{c|c|c|c|c}
A & A+B & A+B+C & A+B+C+D \\
\end{array}
\]
Finding Work Donors and Recipients

- Average load is 5. If \( \pm 2 \), leave alone. Low load \( \leq 3 \). High load \( \geq 7 \).

- Mark donors

0 9 3 4 10 8 2 4

- Compute donor rank with prefix (scan)

0 1 0 0 1 1 0 0

- Similarly, compute receiver rank

1 1 1 1 1 1 1 2 2
Rendezvous

- Allows processors to “meet” each other through an intermediary
- Donor $i$ and receiver $i$ meet at processor $i$. 
Rendezvous

- Allows processors to “meet” each other through an intermediary
- Donor $i$ and receiver $i$ meet at processor $i$. 

4 jobs

4 jobs
Non-Preemptive Scheduler is Sufficient for PICO

- Processes are cooperating
- Control is returned voluntarily so data structures left in clean state
  - No memory access conflicts, no locks
- PICO has its own “thread” scheduler
  - High priority, short threads are round robin and done first
    - Hub communications, incumbent broadcasts, sending subproblems
    - If these are delayed by long tasks could lead to
      - Idle processors
      - Processors working on low-quality work
  - Compute threads are proportional share (stride) scheduling
    - Adjust during computation (e.g. between lower and upper-bounding)
Termination Detection

General issue with asynchronous message-passing programs.

Sense non-overlapping time slices; would like to terminate when
- All processors are idle
- Count of messages sent = count of message received

But this is not sufficient since time slices “wiggle”!

PICO uses the “four (3?) counters” method of Mattern et. al

Sent = Received? Recheck Sent = Received
Checkpointing

- Parallel Jobs (especially MPI-based) crash
  - Machine/routing failure
  - Exceeding time quota
- Checkpointing in PEBBL to save computational state
  - Load balancer message sweep signals checkpoint
  - Workers and hubs go quiet: no new communication
  - Use termination logic to detect safe state
  - Each processor writes checkpoint file
- Restarting
  - Normal: each processor restarts from its own file
  - Read serially and redistribute
    - Allows start up on an arbitrary number of processors
Conclusions

Efficient, scalable parallel branch-and-bound can be nontrivial
- Asynchronous parallelization
- Distributed memory management
- Flexibility of parallel design

Parallel B&B frameworks like PEBBL would be a good base for a parallel MINLP

PEBBL and PICO are part of ACRO
- A Common Repository for Optimizers
  - http://software.sandia.gov/acro
- Self-contained package for parallel MILP (or branch-and-bound)
  - Acro bundles necessary third-party libraries
- BSD license for core libraries
- Acro 2.0 release - December 2008