Accelerating Domain Propagation: an Efficient GPU-Parallel Algorithm over Sparse Matrices

Boro Sofranac [1,2] Ambros Gleixner [1,3] Sebastian Pokutta [1,2]

{sofranac, gleixner, pokutta}@zib.de

[1]Zuse Institute Berlin

[2]Berlin Institute of Technology

[3]HTW Berlin

IOL&COGA Seminar · 10.12.2020

Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

MODAL FORSCHUNGSCAMPUS

ZIB ZUSE INSTITUTE BERLIN

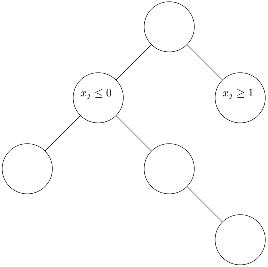
Introduction

Mixed Integer Linear Programming (MIP)

MIP is defined as:

$$
\begin{aligned}
\min \quad & c^T x \\
\text{s. t.} \quad & Ax \leq b \\
& l_j \leq x_j \leq u_j \qquad \forall j \in \mathcal{N} \\
& x_j \in \mathbb{Z} \qquad \forall j \in \mathcal{I}
\end{aligned}
$$

1. Theory: MIPs are $\mathcal{NP}$-hard
2. Practice: surprisingly fast solvers exist
3. Branch-and-bound algorithm most common



Branch-and-bound search tree

Domain Propagation in MIP

Domain propagation: tightens the bounds of variables

$$
\begin{aligned}
&-1 \leq x_1 + x_2 + x_3 \leq 1 \\
&-9 \leq x_1 \leq 9 \\
&-1 \leq x_2 \leq 1 \\
&-1 \leq x_3 \leq 1
\end{aligned}
$$

Domain Propagation in MIP

Domain propagation: tightens the bounds of variables

$$
\begin{aligned}
&-1 \leq x_1 + x_2 + x_3 \leq 1 \\
&-9 \leq x_1 \leq 9 \\
&-1 \leq x_2 \leq 1 \\
&-1 \leq x_3 \leq 1 \\
u_1^{\text{new}}: \; & x_1 \leq 1 - x_2 - x_3 \leq 3
\end{aligned}
$$

Domain Propagation in MIP

Domain propagation: tightens the bounds of variables

$$-1 \leq x_1 + x_2 + x_3 \leq 1$$
$$-9 \leq x_1 \leq 9$$
$$-1 \leq x_2 \leq 1$$
$$-1 \leq x_3 \leq 1$$
$$u_1^{\text{new}} : x_1 \leq 1 - x_2 - x_3 \leq 3$$
$$\ell_1^{\text{new}} : x_1 \geq -1 - x_2 - x_3 \geq -3$$

Domain Propagation in MIP

Domain propagation: tightens the bounds of variables

Generalization:

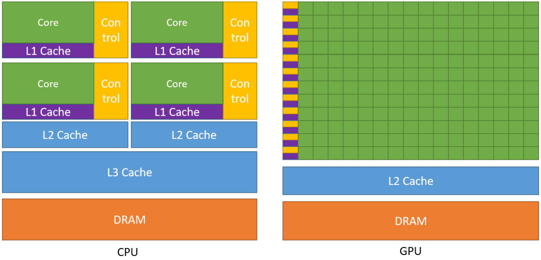
$$\frac{\underline{\beta} - \overline{\alpha}}{a_j} + u_j \leq x_j \leq \frac{\overline{\beta} - \underline{\alpha}}{a_j} + \ell_j, \; a_j > 0$$

$$\frac{\overline{\beta} - \underline{\alpha}}{a_j} + u_j \leq x_j \leq \frac{\underline{\beta} - \overline{\alpha}}{a_j} + \ell_j, \; a_j < 0$$

Iterated domain propagation

1. Can be seen as a fixed-point iteration
2. May not converge in finite time
3. Tolerance-based termination in practice

MIPs and GPUs



Source: `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html`

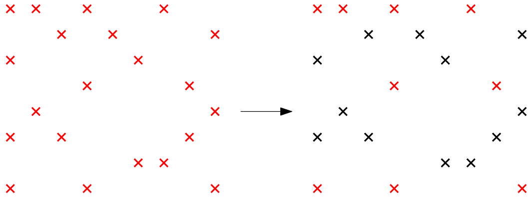
State-of-the-art solvers do not use GPUs. Major challenges:

1. Very irregular and sparse data structures
2. Non-uniform algorithmic behavior

Algorithmic Design

Sequential Propagation Features & GPU Challenges

The constraint marking mechanism



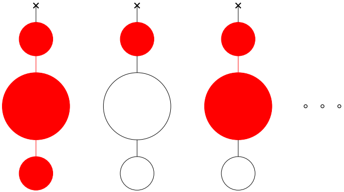
Constraint marking example for two propagation rounds.

GPU implementation challenges:

1. Non-coallesced & random memory accesses
2. Dynamic redistribution of work needed for good load-balancing

Sequential Propagation Features & GPU Challenges

The early termination checks

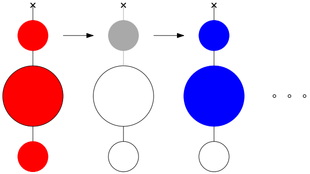


Effect of early termination checks on the workflow of the algorithm

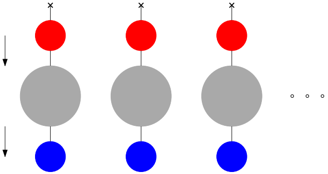
Main GPU implementation challenge:

1. Load-balancing

GPU-parallel Algorithm



Workflow of the sequential algorithm



Workflow of the GPU algorithm

1. Remove features like constraint marking and early termination
2. Switch to throughput-based workflow

Tradeoff: well-structured, static execution at the expense of more work

Handling the Irregular Structure of the Constraint Matrix

Computing min and max activities

Step 1: Compute $\underline{\alpha}$ and $\overline{\alpha}$

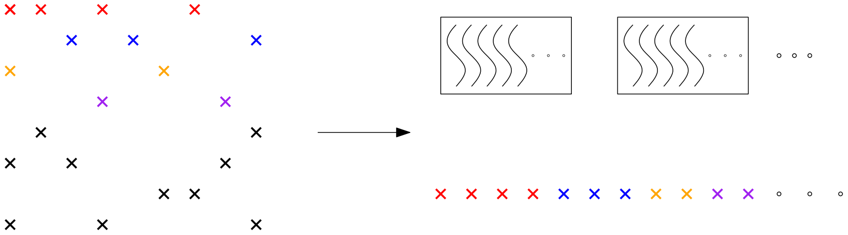
$$\underline{\alpha} := \sum_{i=0}^{n} a_i b_i \text{ with } b_i = \begin{cases} \ell_i & \text{if } a_i > 0, \\ u_i & \text{if } a_i \leq 0, \end{cases} \tag{1a}$$

$$\overline{\alpha} := \sum_{i=0}^{n} a_i b_i \text{ with } b_i = \begin{cases} u_i & \text{if } a_i > 0, \\ \ell_i & \text{if } a_i \leq 0, \end{cases} \tag{1b}$$

- Looks similar to SpMV: Ax
- SpMV is highly bandwidth bound
- Memory: Computing $\underline{\alpha}$ and $\overline{\alpha}$ same as computing Al and Au

Approach: Carry over idea from *CSR-adaptive* [Greathouse and Data 2014] and adapt to $\underline{\alpha}$ and $\overline{\alpha}$.

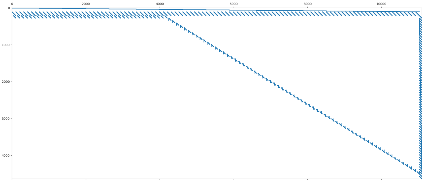
CSR-scalar, CSR-vector, CSR-stream



Access patterns of the CSR values array.

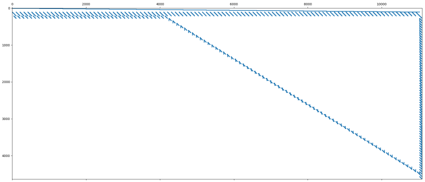
- *CSR-scalar* exhibits poor memory coalescing
- *CSR-vector* exhibits poor load balancing for short rows
- *CSR-stream*: group rows with few non-zeros together

The Domain Propagation Kernel



Sparsity pattern of MIPLIB instance *drayage-25-27*.

The Domain Propagation Kernel



Sparsity pattern of MIPLIB instance *drayage-25-27*.

- *CSR-stream*, *CSR-vector* adapted to compute $\underline{\alpha}$ and $\overline{\alpha}$
- *CSR-vector*: use one warp
- *CSR-vector-long*: use whole block

Domain Propagation Kernel (1 round)

```
if end_row - start_row > 1 then
    CSR-stream
else
    if nnz < threshold then
        CSR-vector
    else
        CSR-vector-long version
compute ℓ_{i,j}^{cand}, u_{i,j}^{cand}
atomic u_j ← min(u_j, u_{i,j}^{cand})
atomic ℓ_j ← max(ℓ_j, ℓ_{i,j}^{cand})
```

Computational Results

Computational Experiments Setup

The test set: MIPLIB 2017

- 1065 MIP instances
- Remove small instances with *variables* < 1000 and *constraints* < 1000
- Divide in 8 subsets of increasing size

Machines:

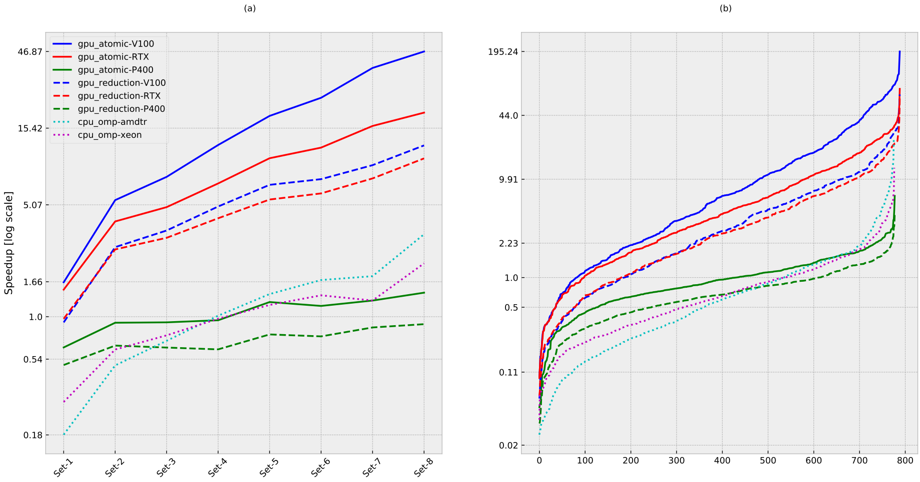
1. **xeon** 24-core Intel Xeon Gold
2. **amdtr** 64-core AMD Ryzen Threadripper
3. **V100** NVIDIA Tesla V100
4. **RTX** NVIDIA Titan RTX
5. **P400** NVIDIA Quadro P400

Algorithms:

1. **cpu_seq**
2. **cpu_omp**
3. **gpu_reduction**
4. **gpu_atomic**

Metric: Speedup over wall-clock time. Base case: **xeon+cpu_seq**

Computational Results



(a) geometric mean of speedups and (b) speedup distributions in ascending order