

Parallelizing the (revised) simplex method

Is it time to give up?

Julian Hall¹
Qi Huangfu¹ Edmund Smith¹ Miles Lubin²

¹School of Mathematics, University of Edinburgh

²Mathematics and Computer Science Division, Argonne National Laboratory

1st August 2013

- Background
 - Linear programming problems
 - The simplex method
 - Historical attempts to exploit parallelism
- Review of recent work
 - Suboptimization
 - Single iteration parallelism
 - Exploiting block structure
 - GPU and multicore standard simplex method
- Is it time to give up?

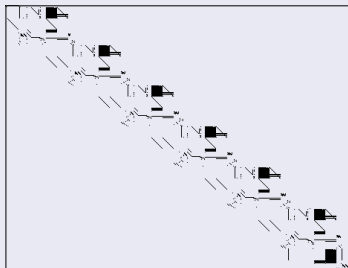
Linear programming (LP)

$$\begin{array}{ll}\text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \quad \mathbf{x} \geq \mathbf{0}\end{array}$$

Background

- Fundamental model in optimal decision-making
- Solution techniques
 - Simplex method (1947)
 - Interior point methods (1984)
- Large problems have
 - 10^3 – 10^7 variables
 - 10^3 – 10^7 constraints
- Matrix A is (usually) sparse

Example



STAIR: 356 rows, 467 columns and 3856 nonzeros

Solving LP problems

$$\begin{array}{ll}\text{minimize} & f_P = \mathbf{c}^T \mathbf{x} \\ \text{subject to} & A\mathbf{x} = \mathbf{b} \quad (P) \\ & \mathbf{x} \geq \mathbf{0}\end{array}$$

$$\begin{array}{ll}\text{maximize} & f_D = \mathbf{b}^T \mathbf{y} \\ \text{subject to} & A^T \mathbf{y} + \mathbf{s} = \mathbf{c} \quad (D) \\ & \mathbf{s} \geq \mathbf{0}\end{array}$$

Optimality conditions

- For a partition $\mathcal{B} \cup \mathcal{N}$ of the variable set with B nonsingular in

$$B\mathbf{x}_B + N\mathbf{x}_N = \mathbf{b} \text{ for (P) and } \begin{bmatrix} B^T \\ N^T \end{bmatrix} \mathbf{y} + \begin{bmatrix} \mathbf{s}_B \\ \mathbf{s}_N \end{bmatrix} = \begin{bmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{bmatrix} \text{ for (D)}$$

with $\mathbf{x}_N = \mathbf{0}$ and $\mathbf{s}_B = \mathbf{0}$

- Primal** basic variables $\mathbf{x}_B$ given by $\hat{\mathbf{b}} = B^{-1}\mathbf{b}$
 - Dual** non-basic variables $\mathbf{s}_N$ given by $\hat{\mathbf{c}}_N^T = \mathbf{c}_N^T - \mathbf{c}_B^T B^{-1}N$
- Partition is optimal if there is
 - Primal feasibility** $\hat{\mathbf{b}} \geq \mathbf{0}$
 - Dual feasibility** $\hat{\mathbf{c}}_N \geq \mathbf{0}$

Simplex algorithm: Each iteration

	$\mathcal{N}$	RHS
$\mathcal{B}$	$\hat{\mathbf{a}}_q$	$\hat{\mathbf{b}}$
	$\hat{a}_{pq}$ $\hat{\mathbf{a}}_p^T$	$\hat{b}_p$
	$\hat{\mathbf{c}}_q$ $\hat{\mathbf{c}}_N^T$	

Primal algorithm: Assume $\hat{\mathbf{b}} \geq \mathbf{0}$ Seek $\hat{\mathbf{c}}_N \geq \mathbf{0}$

Scan $\hat{c}_j, j \in \mathcal{N}$, for a good candidate q to leave $\mathcal{N}$ CHUZY

Scan $\hat{b}_i/\hat{a}_{iq}, i \in \mathcal{B}$, for a good candidate p to leave $\mathcal{B}$ CHUZ

Dual algorithm: Assume $\hat{\mathbf{c}}_N \geq \mathbf{0}$ Seek $\hat{\mathbf{b}} \geq \mathbf{0}$

Scan $\hat{b}_i, i \in \mathcal{B}$, for a good candidate p to leave $\mathcal{B}$ CHUZ

Scan $\hat{c}_j/\hat{a}_{pj}, j \in \mathcal{N}$, for a good candidate q to leave $\mathcal{N}$ CHUZY

Simplex algorithm: Each iteration

	$\mathcal{N}$	RHS
$\mathcal{B}$	$\hat{\mathbf{a}}_q$	$\hat{\mathbf{b}}$
	$\hat{a}_{pq}$ $\hat{\mathbf{a}}_p^T$	$\hat{b}_p$
	$\hat{c}_q$ $\hat{\mathbf{c}}_N^T$	

Update

Exchange p and q between $\mathcal{B}$ and $\mathcal{N}$

Update $\hat{\mathbf{b}} := \hat{\mathbf{b}} - \theta_p \hat{\mathbf{a}}_q$ $\theta_p = \hat{b}_p / \hat{a}_{pq}$ **UPDATE-PRIMAL**

Update $\hat{\mathbf{c}}_N^T := \hat{\mathbf{c}}_N^T - \theta_d \hat{\mathbf{a}}_p^T$ $\theta_d = \hat{c}_q / \hat{a}_{pq}$ **UPDATE-DUAL**

Challenges

- **Algorithmic:** How to identify “good candidates”
- **Computational:** How to obtain $\hat{\mathbf{a}}_p$ and $\hat{\mathbf{a}}_q$

Standard simplex method (SSM): Computation

Major computational component

Update of tableau:

$$\hat{N} := \hat{N} - \frac{1}{\hat{a}_{pq}} \hat{\mathbf{a}}_q \hat{\mathbf{a}}_p^T$$

where $\hat{N} = B^{-1}N$

- Hopelessly inefficient for sparse LP problems
- Prohibitively expensive for large LP problems

Revised simplex method (RSM): Computation

Major computational components

$$\pi_p^T = \mathbf{e}_p^T B^{-1} \quad \text{BTRAN}$$

$$\hat{\mathbf{a}}_q = B^{-1} \mathbf{a}_q \quad \text{FTRAN}$$

$$\hat{\mathbf{a}}_p^T = \pi_p^T N \quad \text{PRICE}$$

$$\text{Invert } B \quad \text{INVERT}$$

Hyper-sparsity

- Vectors $\mathbf{e}_p$ and $\mathbf{a}_q$ are **always sparse**
- B may be **highly reducible**; B^{-1} may be **sparse**
- Vectors π_p , $\hat{\mathbf{a}}_p$ and $\hat{\mathbf{a}}_q$ **may be sparse**
- Efficient implementations must exploit these features
 - H and McKinnon (1998–2005), Bixby (1999)
 - Clp, Koberstein and Suhl (2005–2008)

Exploiting parallelism: Background

Data parallel standard simplex method

- Good parallel efficiency *was* achieved
- Only relevant for dense LP problems

Data parallel revised simplex method

- Only immediate parallelism is in forming $\pi_p^T N$
 - When $n \gg m$ significant speed-up *was* achieved
- Bixby and Martin (2000)

Task parallel revised simplex method

- Overlap computational components for different iterations
Wunderling (1996), H and McKinnon (1995-2005)
- Modest speed-up *was* achieved on general sparse LP problems

Exploiting parallelism: Recent work

- Brief overview of six parallel simplex projects (2008–13)
- Concept of each is relatively straightforward
- Hard work is
 - Making codes efficient and robust in serial
 - Developing efficient and robust parallel implementations

Project 1: Suboptimization (PAMI)

- Little-used dual simplex technique of Roseander (1975)
 - Choose subset of attractive rows
 - Perform minor iterations of dual SSM on these rows
 - Update RSM data
- Dual version of “multiple pricing” for primal simplex algorithm
- Scope for task parallel FTRAN and BTRAN and data parallel PRICE
- Serial advances
 - Novel candidate “quality control” techniques
 - Novel simplex update techniques
 - Novel edge weight pricing techniques
 - Valuable insight into promoting hyper-sparsity
- Performance of serial solver **competitive with c1p**

Huangfu and H (2012–13)

Project 1: Suboptimization (PAMI)

- Results on shared memory four quad-core machine
 - For 30 large test problems: average speed-up 1.5; best 2.63
 - Best for hyper-sparse problems
 - Parallel performance **superior to serial c1p**
- Limits on parallel performance
 - Memory bandwidth
 - Serial computation during INVERT

Project 2: Single iteration parallelism (SIP)

- Exploit limited parallelism in a single dual simplex iteration
 - Data parallel PRICE and CHUZC
 - Task parallel FTRANS: standard, DSE and BFRT
 - Task parallel UPDATE-BASISs: primal, dual and DSE
- More extensive and efficient than work of Forrest (2012)

Huangfu and H (2013)

Project 2: Single iteration parallelism (SIP)

- Results on shared memory four quad-core machine
 - For 30 large test problems: average speed-up 1.1; best 2.05
 - Best for non hyper-sparse problems
- Limits on parallel performance
 - Memory bandwidth
 - Amdahl's law
- Performance of PAMI and SIP is complementary
- Better performance **comprable with serial cplex**

Project 3: Exploiting block structure (1)

minimize $\mathbf{c}^T \mathbf{x}$
subject to $\mathbf{A}\mathbf{x} = \mathbf{b} \quad \mathbf{x} \geq \mathbf{0}$

$$\mathbf{A} = \begin{bmatrix} A_{01} & A_{02} & \dots & A_{0r} \\ & A_{11} & & \\ & & A_{22} & \\ & & & \ddots \\ & & & & A_{rr} \end{bmatrix}$$

- Structure

- The **linking rows** are $[A_{01} \ A_{02} \ \dots \ A_{0r}]$
- The **diagonal blocks** are $[A_{11} \ A_{22} \ \dots \ A_{rr}]$
- Diagonal blocks can be many or few; dense or sparse

- Origin

- Occur naturally in (eg) decentralised planning and multicommodity flow
- BALP structure can be identified in general sparse LPs

Project 3: Exploiting block structure (1)

- Block structure inherited by basis matrix $B = \begin{bmatrix} S & C \\ R & T \end{bmatrix}$ where
 - T is block diagonal
 - Small dimension of S , C and R is number of linking rows
- To solve systems involving B requires operations with
 - factored representation of T
 - matrices C and $\hat{R} = T^{-1}R$
 - factored representation **Schur complement** $W = S - CT^{-1}R$
- Structure yields data parallelism in INVERT, FTRAN and BTRAN
- Structure of N yields (further) data parallelism in PRICE
- All looks good in theory

Project 3: Exploiting block structure (1)

- Primal revised simplex implementation uses
 - Product form updates of individual blocks of T^{-1}
 - Novel rank-2 update of W for stability
 - Many standard algorithmic enhancements
- Performance of serial solver comparable with c1p
- Results on shared memory twin octo-core machine
 - For Netlib problems: average speed-up 0.5; best 1.5
 - For larger test problems: average speed-up 1.0; best 2.0
- Limits on parallel performance
 - Memory bandwidth
 - Exploiting hyper-sparsity
 - Numerical recovery

H and Smith (2010–12)

Project 4: Exploiting block structure (2)

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T \mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{b} \quad \mathbf{x} \geq \mathbf{0} \end{array} \quad \mathbf{A} = \begin{bmatrix} A_{10} & A_{11} & & & \\ A_{20} & & A_{22} & & \\ \vdots & & & \ddots & \\ A_{r0} & & & & A_{rr} \end{bmatrix}$$

- Structure inherited by basis matrix $B = \begin{bmatrix} S & C \\ R & T \end{bmatrix}$ where
 - T is block diagonal of variable dimensions
 - Small dimension of S , C and R is number of linking columns
- Data parallelism in FTRAN, BTRAN, PRICE and INVERT

Project 4: Exploiting block structure (2)

- Dual revised simplex implementation uses
 - Product form updates of B^{-1}
 - Many standard algorithmic enhancements
- Performance of serial solver **comparable with c1p**
- Distributed memory implementation at Argonne on
 - 320-node (2560-core) cluster
 - Blue Gene supercomputer
- Experiments with vast stochastic programming problems
 - 10^6 – 10^8 variables/constraints
 - 70–90% parallel efficiency up to 32 cores
 - 50–70% parallel efficiency up to 128 cores
 - Up to **100 times faster than c1p**

Lubin and H (2012)

Project 5: GPU implementation of SSM and dense RSM

- New architecture $\Rightarrow$ SSM experiments
- Prototype implementations tested on random problems
Spampinato and Elster (2009),
Bieling *et al.* (2010), Lalami *et al.* (2011)
- Stable implementation
 - For Netlib set: 95% success rate
 - For **random dense problems** of increasing size:
10–100 times faster than cplex/gurobi
H and Smith (2012)

Project 6: Multicore implementation of SSM

Stable implementation on dual quad-core machine

- For Netlib set: 90% success rate
- For **random dense problems** of increasing size
 - 10–2 times speed-up
 - **40–10 times faster than cplex/gurobi**
- Cache effect for smaller problems
- Memory bound for larger problems

H and Smith (2012)

Parallelizing the simplex method: Is it time to give up?

No “silver bullet” for general parallel simplex

Good parallel performance *is possible*

- For small LPs distributed across multi-core cache
- For structured LPs distributed on supercomputer
- For dense LPs on GPU

Is this problem/method/architecture environment relevant?

Parallel research has yielded serial results

- Novel update techniques
- Insight into promoting hyper-sparsity
- Novel edge weight strategies
- Huangfu's c++ solver hsol

See <http://www.maths.ed.ac.uk/hall/ICCOPT13>

References



J. A. J. Hall and Q. Huangfu.

A high performance dual revised simplex solver.

In R. W. et al., editor, *PPAM 2011, Part I*, volume 7203 of *LNCS*, pages 143–151, Heidelberg, 2012. Springer.



Q. Huangfu and J. A. J. Hall.

Novel update techniques for the revised simplex method.

Technical Report ERGO-13-001, School of Mathematics, University of Edinburgh, 2013. Submitted to *Computational Optimization and Applications*.



Q. Huangfu and J. A. J. Hall.

Parallelizing the dual revised simplex method with suboptimization.

Technical report, School of Mathematics, University of Edinburgh, 2013. in preparation.



M. Lubin, J. A. J. Hall, C. G. Petra, and M. Anitescu.

Parallel distributed-memory simplex for large-scale stochastic LP problems.

Computational Optimization and Applications, 55(3):571–596, 2013.