THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

A high performance dual revised simplex solver

Citation for published version:

Hall, J & Huangfu, Q 2012, 'A high performance dual revised simplex solver', *Lecture Notes in Computer Science*, vol. 7203, pp. 143-151. <http://www.maths.ed.ac.uk/hall/PPAM_DualSuboptimization/ERGO-11-007.pdf>

Link:

[Link to publication record in Edinburgh Research Explorer](#)

Document Version:

Peer reviewed version

Published In:

Lecture Notes in Computer Science

Publisher Rights Statement:

The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-642-31464-3_15

General rights

Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy

The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

A high performance dual revised simplex solver

Julian Hall, Qi Huangfu and Edmund Smith

School of Mathematics

University of Edinburgh

2nd December 2010

A high performance dual revised simplex solver

Overview

- LP problems and the dual simplex method
- Why use dual simplex method
- Why exploit parallelism
- Dual revised simplex with suboptimization
- Preliminary results
- Alternative product form update for GPU extension
- Conclusions

Linear programming (LP)

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

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

STAIR: 356 rows, 467 columns and 3856 nonzeros

Mathematics of LP

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

- Geometry:

- Feasible points form a convex polyhedron

- Results:

- An optimal solution occurs at a vertex
- At a vertex the variable set can be partitioned as $\mathcal{B} \cup \mathcal{N}$ and constraints as

$$\mathbf{B}\mathbf{x}_B + \mathbf{N}\mathbf{x}_N = \mathbf{b}$$

so $\mathbf{B}$ is nonsingular and $\mathbf{x}_N = \mathbf{0}$

- Dual LP problem:

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

- Result:

- Optimal partition $\mathcal{B} \cup \mathcal{N}$ for (P) also solves (D)

The reduced LP problem

At a vertex, for a partition $\mathcal{B} \cup \mathcal{N}$ with B nonsingular and $\mathbf{x}_N = \mathbf{0}$, the original problem is

$$\begin{aligned} \text{minimize } f &= \mathbf{c}_N^T \mathbf{x}_N + \mathbf{c}_B^T \mathbf{x}_B \\ \text{subject to } N \mathbf{x}_N + B \mathbf{x}_B &= \mathbf{b} \\ \mathbf{x}_N \geq \mathbf{0} \quad \mathbf{x}_B &\geq \mathbf{0}. \end{aligned}$$

Eliminate $\mathbf{x}_B$ from the objective to give the **reduced LP problem**

$$\begin{aligned} \text{minimize } f &= \mathbf{s}_N^T \mathbf{x}_N + \hat{f} \\ \text{subject to } \hat{N} \mathbf{x}_N + I \mathbf{x}_B &= \hat{\mathbf{b}} \\ \mathbf{x}_N \geq \mathbf{0} \quad \mathbf{x}_B &\geq \mathbf{0}, \end{aligned}$$

where $\hat{\mathbf{b}} = B^{-1}\mathbf{b}$, $\hat{N} = B^{-1}N$, $\hat{f} = \mathbf{c}_B^T \hat{\mathbf{b}}$ and $\mathbf{s}_N$ is given by

$$\mathbf{s}_N^T = \mathbf{c}_N^T - \mathbf{c}_B^T \hat{N}$$

Vertex is optimal $\iff \mathbf{x}_B \geq \mathbf{0}$ and $\mathbf{s}_N \geq \mathbf{0}$

Primal vs dual simplex

Finding an optimal partition $\mathcal{B} \cup \mathcal{N}$ underpins the simplex method

- **Primal** simplex method
 - Maintains $\mathbf{x}_B \geq \mathbf{0}$
 - Moves along edges of the feasible region of (P)
 - Terminates when $\mathbf{s}_N \geq \mathbf{0}$
- **Dual** simplex method
 - Maintains $\mathbf{s}_N \geq \mathbf{0}$
 - Moves along edges of the feasible region of (D)
 - Terminates when $\mathbf{x}_B \geq \mathbf{0}$
- Adaptations of both are required to find initial feasible point

Implementation: dual standard simplex method

	$\mathcal{N}$	$\mathcal{B}$	RHS
1 : m	$\hat{N}$	I	$\hat{b}$
0	s_N^T	0^T	$-\hat{f}$

In each iteration:

- Choose a row p with $\hat{b}_p < 0$
- Use the **pivotal row** p of $\hat{N}$ and s_N to find the **pivotal column** q with $\beta = s_q / \hat{a}_{pq}$
- Exchange indices p and q between $\mathcal{B}$ and $\mathcal{N}$
- Update tableau corresponding to this **basis change**

$$\begin{aligned}
 \hat{N} &:= \hat{N} - (1/\hat{a}_{pq})\hat{a}_q\hat{a}_p^T & \hat{b} &:= \hat{b} - (\hat{b}_p/\hat{a}_{pq})\hat{a}_q \\
 s_N^T &:= s_N^T - \beta\hat{a}_p^T & -\hat{f} &:= -\hat{f} - \beta\hat{b}_p
 \end{aligned}$$

Implementation: dual revised simplex method

- Maintains a representation of B^{-1} (rather than explicit $\hat{N}$)
- Rows and columns of $\hat{N}$ obtained as required
- Pivotal row is

$$\hat{\mathbf{a}}_p^T = \boldsymbol{\pi}_p^T \mathbf{N}, \text{ where } \boldsymbol{\pi}_p^T = \mathbf{e}_p^T \mathbf{B}^{-1}$$

- Pivotal column is

$$\hat{\mathbf{a}}_q = \mathbf{B}^{-1} \mathbf{a}_q, \text{ where } \mathbf{a}_q \text{ is column } q \text{ of } \mathbf{A}$$

- Representation of B^{-1} is updated by exploiting

$$\mathbf{B} := \mathbf{B} + (\mathbf{a}_q - \mathbf{a}_p) \mathbf{e}_p^T$$

- Periodically a representation of B^{-1} is formed from scratch
- Efficient solution of large sparse LP problems requires the revised simplex method

Implementation: representing B^{-1}

- Factorize some basis matrix $B_0 = L_0 U_0$
Store non-trivial columns of L_0 and U_0 as representation of B_0^{-1}
- Updating representation of B^{-1} each iteration exploits

$$\begin{aligned} B &:= B + (\mathbf{a}_q - \mathbf{a}_p) \mathbf{e}_p^T \\ &= B[I + B^{-1}(\mathbf{a}_q - \mathbf{a}_p) \mathbf{e}_p^T] \\ &= B[I + (\hat{\mathbf{a}}_q - \mathbf{e}_p) \mathbf{e}_p^T] \end{aligned}$$

where $\hat{\mathbf{a}}_q$ is the pivotal column so, using Sherman-Morrison and $\hat{a}_{pq} = \mathbf{e}_p^T \hat{\mathbf{a}}_q$,

$$B_k^{-1} = \left[I - \frac{(\hat{\mathbf{a}}_q - \mathbf{e}_p) \mathbf{e}_p^T}{\hat{a}_{pq}} \right] B_{k-1}^{-1} = E_k^{-1} B_{k-1}^{-1}$$

- Hence

$$B_k^{-1} = H_k^{-1} B_0^{-1} \quad \text{where} \quad H_k^{-1} = E_k^{-1} \dots E_1^{-1}$$

Summary: major computational components for simplex implementations

Standard simplex method (SSM)

- Update tableau $\hat{N} := \hat{N} - (1/\hat{a}_{pq})\hat{\mathbf{a}}_q\hat{\mathbf{a}}_p^T$

Revised simplex method (RSM)

- Operations
 - Form $\boldsymbol{\pi}_p^T = \mathbf{e}_p^T B^{-1}$
 - Form $\hat{\mathbf{a}}_p^T = \boldsymbol{\pi}_p^T N$
 - Form $\hat{\mathbf{a}}_q = B^{-1}\mathbf{a}_q$
- Inversion of B
- Distinctive features
 - Vectors $\mathbf{e}_p$, $\mathbf{a}_q$ are **always sparse**
 - B may be **highly reducible**
 - B^{-1} **may be sparse**
 - Vectors $\boldsymbol{\pi}_p$, $\hat{\mathbf{a}}_p$ and $\hat{\mathbf{a}}_q$ **may be sparse**
- Efficient implementations must exploit these features
H and McKinnon (1998–2005)

Why use the dual simplex method?

- Dual simplex method has been around almost as long as the primal simplex method
- More of theoretical interest until 1990's
- Now preferred to primal simplex
 - Easier to find feasible point of (D) to start
 - Has some efficient algorithmic tricks not available to primal
 - Dual feasibility retained when constraints are added (MIP)
- Primal simplex method applied to (D) can be made computationally equivalent

CPLEX LP solvers applied to standard test problems

- Dual simplex better than primal but barrier clearly better

CPLEX LP solvers applied to standard test problems

- Little to choose between dual simplex and barrier with crossover

Parallel simplex: why?

- Moore's law drives core counts per processor, but clock speeds will stabilise
- Serial performance of simplex is spectacularly good
 - Flop count per iteration is near optimal
 - Number of iterations is near optimal
- Can't wait for faster serial processors or algorithmic improvement
- Simplex method must try to exploit parallelism

Parallel simplex: immediate scope

- Standard simplex method
 - Update tableau $\hat{N} := \hat{N} - (1/\hat{a}_{pq})\hat{\mathbf{a}}_q\hat{\mathbf{a}}_p^T$
Level 2 BLAS with $\hat{N}$ dense so “massively parallel”
- Revised simplex method
 - Operations $\boldsymbol{\pi}_p^T = \mathbf{e}_p^T B^{-1}$ and $\hat{\mathbf{a}}_q = B^{-1}\mathbf{a}_q$ are “inherently serial”
 - Operation $\hat{\mathbf{a}}_p^T = \boldsymbol{\pi}_p^T N$ is “massively parallel”
Amdahl’s law implies little immediate scope for exploiting parallelism

Parallel simplex: nature of B_0^{-1} and H^{-1}

Properties of B_0^{-1} and H^{-1} are a major influence on parallel schemes

- B_0^{-1} requires **many** vectors but they are **short** $O(1)$
 - Parallelism is fine grained
 - Unsuitable for data parallelism
 - Aim for task parallelism
- H^{-1} requires **few** vectors but they are **long** $O(m)$
 - Operations with H^{-1} can be posed as a sparse matrix-vector product
 - Suitable for data parallelism

Parallel simplex: past work

- **Data parallel standard simplex method**
 - Good parallel efficiency *was* achieved
 - Totally uncompetitive with *serial* revised simplex method without prohibitive resources
- **Data parallel revised simplex method**
 - Only immediate parallelism is in forming $\pi_p^T N$
 - When $n \gg m$, cost of $\pi_p^T N$ dominates: 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
- All data and task parallel implementations compromised by serial inversion of B

Review: H (2010)

Architectures: CPU or GPU or both?

Heterogeneous desk-top architectures

CPU:

- Fewer, faster cores
- Relatively slow memory transfer
- Welcomes algorithmically complex code
- Full range of development tools

GPU:

- More, slower cores
- Relatively fast memory transfer
- Global communication is expensive/difficult
- Very limited development tools

CPU and GPU:

- Possibly combine CPU and GPU to harness full computing power

Scope for task and data parallelism via suboptimization

- Perform multiple pricing—standard simplex **suboptimization**
 - Primal: Orchard-Hays (1968)
 - **Dual**: Rosander (1975)
- Algorithmically
 - Primal: Identify attractive column slice of tableau
 - **Dual**: Identify attractive **row** slice of tableau
 - Both perform standard simplex iterations to identify a set of basis changes
- Computationally
 - Solve systems with multiple RHS
 - Update tableaux
 - Form matrix products with multiple vectors
- Attractive in the days when memory access was expensive...

Primal: Parallel implementations by Wunderling (1996), H and McKinnon (1995-2005)

Dual: New, even in serial?

Dual revised simplex method with suboptimization

Let the current basis be B_k

- Choose attractive row set $\mathcal{P}$
- Form rows $\mathcal{P}$ of B_k^{-1} : $\hat{\pi}_{\mathcal{P}}^T = e_{\mathcal{P}}^T B_k^{-1}$
- Form rows $\mathcal{P}$ of tableau: $\hat{a}_{\mathcal{P}}^T = \pi_{\mathcal{P}}^T N_k$
- Perform l dual standard simplex iterations on $\hat{a}_{\mathcal{P}}^T$ to identify column set $\mathcal{Q}$
- Form pivotal columns for $q \in \mathcal{Q}$: $\hat{a}_{\mathcal{Q}} = B_k^{-1} a_{\mathcal{Q}}$
- Use $\hat{a}_{\mathcal{Q}}$ to update $\hat{b}_k$ and H_k^{-1} to obtain $\hat{b}_{k+l}$ and representation of B_{k+l}^{-1}

Parallel operations with B_k^{-1}

- Consider operations with $B_k^{-1} = H_k^{-1} B_0^{-1}$ in stages corresponding to B_0 and H_k
- Form $\hat{\pi}_{\mathcal{P}}^T = e_{\mathcal{P}}^T B_k^{-1}$ as

$$\tilde{\pi}_{\mathcal{P}}^T = e_{\mathcal{P}}^T H_k^{-1}$$

then

$$\hat{\pi}_{\mathcal{P}}^T = \tilde{\pi}_{\mathcal{P}}^T B_0^{-1}$$

- Form $\hat{a}_{\mathcal{Q}} = B_k^{-1} a_{\mathcal{Q}}$ as

$$\tilde{a}_{\mathcal{Q}} = B_0^{-1} a_{\mathcal{Q}}$$

then

$$\hat{a}_{\mathcal{Q}} = H_k^{-1} \tilde{a}_{\mathcal{Q}}$$

Gantt chart of computation and data

Operation with H_k^{-1}

Operation with B_0^{-1}

Operation with N_k

Operation with $\hat{a}_{\mathcal{P}}^T$

Operation with $\hat{a}_{\mathcal{Q}}$

Operation with b_k

- Inversion of B_k can take a whole major iteration...
 - ... but not longer—for reasons of numerical stability
 - Serial inversion could lead to load imbalance
 - Inversion could be parallel?

Prototype implementation

- Written by Qi Huangfu (2010)
- Uses (generally) highly efficient core routines
 - Hyper-sparse matrix-vector product $\pi_{\mathcal{P}}^T N_k$
 - Dual steepest edge pricing
- Does not (yet) use
 - Hyper-sparse operations with B^{-1}
 - Bound-flipping ratio test
- Written in C++ with OpenMP using the Intel C++ compiler
- Tested on a dual quad-core AMD Opteron 2378 system
- Uses one pivotal row per core used
- First run last week!

Preliminary results

pds-06: 9882 rows, 28655 columns and 82269 nonzeros

	Cores				clp dual
	1	2	4	8	
Major iterations	10266	5049	2543	1253	9808
Total iterations	10266	9625	8820	7616	9808
Solution time (s)	3.76	2.51	2.00	1.52	1.92
Speed (iter/s)	2730	3836	4419	5017	5111

Preliminary results

pds-10: 16559 rows, 48763 columns and 140063 nonzeros

	Cores				clp dual
	1	2	4	8	
Major iterations	17983	9158	4807	2557	17713
Total iterations	17983	17051	16263	15404	17713
Solution time (s)	12.58	10.01	6.86	5.72	6.61
Speed (iter/s)	1430	1704	2370	2695	2682

Can the simplex method exploit a GPU?

- Standard simplex method implemented in single precision by Smith as **i8**
- Experiments on machine with two quad-core AMD CPUs and GTX285 NVIDIA GPU
- For **dense** LP problems: **best results**

Solver	Type	HPC	Time	Iterations	Speed (iter/s)
gurobi	primal RSM	serial	1357	16034	12
gurobi	dual RSM	serial	976	14518	15
i6	primal SSM	parallel	4039	288419	79
i8	primal SSM	GPU	800	221157	276

- Not (really) of practical value but a good learning exercise
- No hope of beating serial solvers on sparse LP problems
- Now adding steepest edge and extending to double precision for Tesla C2070

GPU extension

- Put dual standard simplex iterations on a GPU
- Consider putting other simple “rectangular” operations on a GPU
 - Sparse matrix-vector product $\pi_{\mathcal{P}}^T N_k$
GPU out-performs comparable CPU for full vector (eg) Vuduc *et al.* (2010)
Nvidia interested in case of sparse vector
 - Operations with H_k^{-1}
Can be posed as a sparse matrix-vector product
- Need to limit data transfer between main memory and GPU

Splitting work between CPU and GPU

- CPU and GPU operations can be overlapped
- Data traffic between main memory and GPU is high

Overlapping CPU/GPU computation and communication

- Lucky if overlapping computation accommodates communication

Alternative product form update

- Little-known (unknown?) alternative product form update may offer a solution
- Updating representation of B^{-1} each iteration exploits

$$\begin{aligned} B &:= B + (\mathbf{a}_q - \mathbf{a}_p)\mathbf{e}_p^T \\ &= [I + (\mathbf{a}_q - \mathbf{a}_p)\mathbf{e}_p^T B^{-1}]B \\ &= [I + (\mathbf{a}_q - \mathbf{a}_p)\boldsymbol{\pi}_p^T]B \end{aligned}$$

so, using Sherman-Morrison,

$$B_k^{-1} = B_{k-1}^{-1} \left[I - \frac{(\mathbf{a}_q - \mathbf{a}_p)\boldsymbol{\pi}_p^T}{\hat{a}_{pq}} \right] = B_{k-1}^{-1} E_k^{-1}$$

- Hence **reversed the order of inverse and update** in representation of B_k^{-1}

$$B_k^{-1} = B_0^{-1} H_k^{-1} \quad \text{where} \quad H_k^{-1} = E_1^{-1} \dots E_k^{-1}$$

Alternative dual revised simplex method with suboptimization

- Parallel operations with $B_k^{-1} = B_0^{-1}H_k^{-1}$
- Form $\hat{\pi}_{\mathcal{P}}^T = e_{\mathcal{P}}^T B_k^{-1}$ as

$$\tilde{\pi}_{\mathcal{P}}^T = e_{\mathcal{P}}^T B_0^{-1}$$

then

$$\hat{\pi}_{\mathcal{P}}^T = \tilde{\pi}_{\mathcal{P}}^T H_k^{-1}$$

- Form $\hat{a}_{\mathcal{Q}} = B_k^{-1} a_{\mathcal{Q}}$ as

$$\tilde{a}_{\mathcal{Q}} = H_k^{-1} a_{\mathcal{Q}}$$

then

$$\hat{a}_{\mathcal{Q}} = B_0^{-1} \tilde{a}_{\mathcal{Q}}$$

Splitting work between CPU and GPU

- Data traffic between main memory and GPU is lower
- Shorter time for inversion of B_{k+l}

Overlapping CPU/GPU computation and communication

- Greater scope for overlapping computation
- Implementation will be **very** difficult

Conclusions

- Identified need for simplex method to exploit parallelism
- Developed prototype high performance dual revised simplex solver
- Initial results are encouraging
- Shown that the standard simplex method will run fast on a GPU
- Standard approach to linear algebra bad for CPU-GPU combination
- Alternative product form update may offer a solution

References

- [1] R. E. Bixby and A. Martin. Parallelizing the dual simplex method. *INFORMS Journal on Computing*, 12:45–56, 2000.
- [2] J. A. J. Hall. Towards a practical parallelisation of the simplex method. *Computational Management Science*, 7(2):139–170, 2010.
- [3] J. A. J. Hall and K. I. M. McKinnon. PARSMI, a parallel revised simplex algorithm incorporating minor iterations and Devex pricing. In J. Waśniewski, J. Dongarra, K. Madsen, and D. Olesen, editors, *Applied Parallel Computing*, volume 1184 of *Lecture Notes in Computer Science*, pages 67–76. Springer, 1996.
- [4] J. A. J. Hall and K. I. M. McKinnon. ASYNPLEX, an asynchronous parallel revised simplex method algorithm. *Annals of Operations Research*, 81:27–49, 1998.
- [5] J. A. J. Hall and K. I. M. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32(3):259–283, December 2005.
- [6] W. Orchard-Hays. *Advanced Linear programming computing techniques*. McGraw-Hill, New York, 1968.

- [7] R. R. Rosander. Multiple pricing and suboptimization in dual linear programming algorithms. *Mathematical Programming Study*, 4:108–117, 1975.
- [8] R. Vuduc, A. Chandramowlishwarany, J. Choi, M. Guney, and A. Shringarpure. On the limits of GPU acceleration. *Not known*, 2010.
- [9] R. Wunderling. Paralleler und objektorientierter simplex. Technical Report TR-96-09, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1996.