Scalable massively parallel simplex algorithms for block-structured linear programs

İlkay Boduroğlu[1], Julian Hall[2] and Jonathan Hogg[2]

[1]Informatics Institute, Istanbul Technical University

[2]School of Mathematics, University of Edinburgh

September 14th 2007

Simplex methods for LP problems

$$
\begin{array}{rl}
\text{minimize} & f = \boldsymbol{c}^T \boldsymbol{x} \\
\text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\
 & \boldsymbol{x} \geq \boldsymbol{0} \\
\text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m
\end{array}
$$

Simplex methods for LP problems

$$
\begin{array}{rl}
\text{minimize} & f = \boldsymbol{c}^T \boldsymbol{x} \\
\text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\
 & \boldsymbol{x} \geq \boldsymbol{0} \\
\text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m
\end{array}
$$

- If A is dense use the simplex method with dense matrix algebra

Simplex methods for LP problems

$$
\begin{array}{rl}
\text{minimize} & f = \boldsymbol{c}^T \boldsymbol{x} \\
\text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\
 & \boldsymbol{x} \geq \boldsymbol{0} \\
\text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m
\end{array}
$$

- If A is dense use the simplex method with dense matrix algebra
- If A is sparse but unstructured use a sparsity-exploiting revised simplex solver

Simplex methods for LP problems

$$\begin{array}{rl} \text{minimize} & f = \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ \text{where} & \boldsymbol{x} \in I\!R^n \quad \text{and} \quad \boldsymbol{b} \in I\!R^m \end{array}$$

- If A is dense use the simplex method with dense matrix algebra
- If A is sparse but unstructured use a sparsity-exploiting revised simplex solver
- If A is has block-angular form

$$\begin{bmatrix} A_0 & A_1 & A_2 & \dots & A_r \\ & D_1 & & & \\ & & D_2 & & \\ & & & \ddots & \\ & & & & D_r \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} A_0 & & & & \\ A_1 & D_1 & & & \\ A_2 & & D_2 & & \\ \vdots & & & \ddots & \\ A_r & & & & D_r \end{bmatrix}$$

consider methods that exploit this structure.

Parallel simplex

Outcome of work in this area is reviewed by Hall (2006) and summarised as follows

- When arithmetic is dense
  - Parallelisation is readily achieved

Parallel simplex

Outcome of work in this area is reviewed by Hall (2006) and summarised as follows

- When arithmetic is dense
  - Parallelisation is readily achieved
  - For sparse LP problems parallel dense solvers are uncompetitive with good serial solvers

Parallel simplex

Outcome of work in this area is reviewed by Hall (2006) and summarised as follows

- When arithmetic is dense
  - Parallelisation is readily achieved
  - For sparse LP problems parallel dense solvers are uncompetitive with good serial solvers unless implementation is *massively parallel*

Parallel simplex

Outcome of work in this area is reviewed by Hall (2006) and summarised as follows

- When arithmetic is dense
  - Parallelisation is readily achieved
  - For sparse LP problems parallel dense solvers are uncompetitive with good serial solvers unless implementation is *massively parallel*
- When arithmetic is sparse
  - Parallelisation is *very hard* to achieve

Parallel simplex

Outcome of work in this area is reviewed by Hall (2006) and summarised as follows

- When arithmetic is dense
  - Parallelisation is readily achieved
  - For sparse LP problems parallel dense solvers are uncompetitive with good serial solvers unless implementation is *massively parallel*
- When arithmetic is sparse
  - Parallelisation is *very hard* to achieve
  - Any parallel performance is worthwhile

Parallel simplex

Outcome of work in this area is reviewed by Hall (2006) and summarised as follows

- When arithmetic is dense
  - Parallelisation is readily achieved
  - For sparse LP problems parallel dense solvers are uncompetitive with good serial solvers unless implementation is *massively parallel*
- When arithmetic is sparse
  - Parallelisation is *very hard* to achieve
  - Any parallel performance is worthwhile
- However methods for block-angular LPs using dense matrix algebra offer scope for worthwhile parallel solvers

Massively parallel standard simplex method

- Dominant computational cost is updating the tableau using outer product operations

$$\hat{A} := \hat{A} + \boldsymbol{u}\boldsymbol{v}^T$$

- Maps easily onto shared memory architecture using dense data structures and BLAS

Massively parallel standard simplex method

- Dominant computational cost is updating the tableau using outer product operations

$$\hat{A} := \hat{A} + \boldsymbol{u}\boldsymbol{v}^T$$

- Maps easily onto shared memory architecture using dense data structures and BLAS
- Notable massively parallel implementation on Connection Machine CM-2
  Eckstein *et al.* (inc. Boduroğlu): ORSA J. Comput. **7(4)** (1995) 402–416

Massively parallel standard simplex method

- Dominant computational cost is updating the tableau using outer product operations

$$\hat{A} := \hat{A} + \boldsymbol{u}\boldsymbol{v}^T$$

- Maps easily onto shared memory architecture using dense data structures and BLAS
- Notable massively parallel implementation on Connection Machine CM-2
  Eckstein *et al.* (inc. Boduroğlu): ORSA J. Comput. **7(4)** (1995) 402–416
- Incorporated
  - Steepest edge pricing: Goldfarb and Reid, MP **12** (1977) 361–371
  - EXPAND ratio test: Gill *et al.* , MP **45** (1989) 437–474

Massively parallel standard simplex method

- Dominant computational cost is updating the tableau using outer product operations

$$\hat{A} := \hat{A} + \boldsymbol{u}\boldsymbol{v}^T$$

- Maps easily onto shared memory architecture using dense data structures and BLAS
- Notable massively parallel implementation on Connection Machine CM-2
  Eckstein *et al.* (inc. Boduroğlu): ORSA J. Comput. **7(4)** (1995) 402–416
- Incorporated
  - Steepest edge pricing: Goldfarb and Reid, MP **12** (1977) 361–371
  - EXPAND ratio test: Gill *et al.* , MP **45** (1989) 437–474
- Achieved
  - linear speed-up
  - Remarkable numerical stability
- Applied to data mining problems

Methods for block-angular LP problems

- Dantzig-Wolfe and Benders decomposition are traditional approaches for, respectively,

$$\begin{bmatrix} A_0 & A_1 & A_2 & \dots & A_r \\ & D_1 & & & \\ & & D_2 & & \\ & & & \ddots & \\ & & & & D_r \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} A_0 & & & & \\ A_1 & D_1 & & & \\ A_2 & & D_2 & & \\ \vdots & & & \ddots & \\ A_r & & & & D_r \end{bmatrix}$$

Methods for block-angular LP problems

- Dantzig-Wolfe and Benders decomposition are traditional approaches for, respectively,

$$\begin{bmatrix} A_0 & A_1 & A_2 & \dots & A_r \\ & D_1 & & & \\ & & D_2 & & \\ & & & \ddots & \\ & & & & D_r \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} A_0 & & & & \\ A_1 & D_1 & & & \\ A_2 & & D_2 & & \\ \vdots & & & \ddots & \\ A_r & & & & D_r \end{bmatrix}$$

- Both have been parallelised but cannot incorporate edge-weight pricing strategies for efficiency and numerical stability

Methods for block-angular LP problems

- Dantzig-Wolfe and Benders decomposition are traditional approaches for, respectively,

$$\begin{bmatrix} A_0 & A_1 & A_2 & \dots & A_r \\ & D_1 & & & \\ & & D_2 & & \\ & & & \ddots & \\ & & & & D_r \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} A_0 & & & & \\ A_1 & D_1 & & & \\ A_2 & & D_2 & & \\ \vdots & & & \ddots & \\ A_r & & & & D_r \end{bmatrix}$$

- Both have been parallelised but cannot incorporate edge-weight pricing strategies for efficiency and numerical stability
- Kaul's algorithm (1965) performs standard simplex iterations exploiting (row-linked) block-angular structure
  - Edge-weight pricing strategies can be incorporated
  - Dense arithmetic is restricted to "small" matrices

Kaul's algorithm: introduction

- When A is in block angular form and partitioned as $[B\ N]$, the **basis matrix** B has structure

Kaul's algorithm: introduction

- When A is in block angular form and partitioned as $[B\ N]$, the **basis matrix** B has structure

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad \text{where} \qquad T = \begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_r \end{bmatrix}$$

Kaul's algorithm: introduction

- When A is in block angular form and partitioned as $[B\ N]$, the **basis matrix** B has structure

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad \text{where} \qquad T = \begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_r \end{bmatrix}$$

- The standard simplex tableau can be represented as

$$B^{-1}N = B^{-1}\begin{bmatrix} N_1 \\ N_2 \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}$$

  where
  - Z_1 contains the tableau rows corresponding to the linking rows
  - Z_2 can be expressed as $Z_2 = VZ_1 + \Lambda N_2$ where $V = -\Lambda R$ and $\Lambda = T^{-1}$

Kaul's algorithm: introduction

- When A is in block angular form and partitioned as $[B\ N]$, the **basis matrix** B has structure

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad \text{where} \qquad T = \begin{bmatrix} B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_r \end{bmatrix}$$

- The standard simplex tableau can be represented as

$$B^{-1}N = B^{-1}\begin{bmatrix} N_1 \\ N_2 \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}$$

  where
  - Z_1 contains the tableau rows corresponding to the linking rows
  - Z_2 can be expressed as $Z_2 = VZ_1 + \Lambda N_2$ where $V = -\Lambda R$ and $\Lambda = T^{-1}$
- Simplex iterations are performed by maintaining Z_1, V and $\Lambda_i = B_i^{-1}$, for $i = 1, \ldots, r$
  - Matrices Z_1 and V have one "small" dimension
  - Each Λ_i is a "small" square matrix

Kaul's algorithm: iterations

Using the tableau representation

$$\begin{bmatrix} Z_1 \\ VZ_1 + \Lambda N_2 \end{bmatrix}$$

The data for each simplex iteration are obtained as follows

Kaul's algorithm: iterations

Using the tableau representation

$$\begin{bmatrix} Z_1 \\ VZ_1 + \Lambda N_2 \end{bmatrix}$$

The data for each simplex iteration are obtained as follows

- Let the (pivotal) column q of the tableau correspond to diagonal block i

Kaul's algorithm: iterations

Using the tableau representation

$$\begin{bmatrix} Z_1 \\ VZ_1 + \Lambda N_2 \end{bmatrix}$$

The data for each simplex iteration are obtained as follows

- Let the (pivotal) column q of the tableau correspond to diagonal block i
  - Components from Z_1 are known explicitly
  - Components from Z_2 are formed using V, column q of Z_1 and a product with Λ_i

Kaul's algorithm: iterations

Using the tableau representation

$$\begin{bmatrix} Z_1 \\ VZ_1 + \Lambda N_2 \end{bmatrix}$$

The data for each simplex iteration are obtained as follows

- Let the (pivotal) column q of the tableau correspond to diagonal block i
  - Components from Z_1 are known explicitly
  - Components from Z_2 are formed using V, column q of Z_1 and a product with Λ_i
- For the (pivotal) row p of the tableau
  - If p is a linking row then the tableau row is known explicitly as a row of Z_1

Kaul's algorithm: iterations

Using the tableau representation

$$\begin{bmatrix} Z_1 \\ VZ_1 + \Lambda N_2 \end{bmatrix}$$

The data for each simplex iteration are obtained as follows

- Let the (pivotal) column q of the tableau correspond to diagonal block i
  - Components from Z_1 are known explicitly
  - Components from Z_2 are formed using V, column q of Z_1 and a product with Λ_i
- For the (pivotal) row p of the tableau
  - If p is a linking row then the tableau row is known explicitly as a row of Z_1
  - Otherwise, p corresponds to diagonal block i and the tableau row is formed using a row of V, Z_1 and a product with Λ_i^T

Kaul's algorithm: iterations

Using the tableau representation

$$\begin{bmatrix} Z_1 \\ VZ_1 + \Lambda N_2 \end{bmatrix}$$

The data for each simplex iteration are obtained as follows

- Let the (pivotal) column q of the tableau correspond to diagonal block i
  - Components from Z_1 are known explicitly
  - Components from Z_2 are formed using V, column q of Z_1 and a product with Λ_i
- For the (pivotal) row p of the tableau
  - If p is a linking row then the tableau row is known explicitly as a row of Z_1
  - Otherwise, p corresponds to diagonal block i and the tableau row is formed using a row of V, Z_1 and a product with Λ_i^T
- The updates of Z_1, V and (the appropriate) Λ_i require dense outer-product operations

Kaul's algorithm: approximate steepest edge pricing

- Implementation of Kaul's algorithm uses approximate steepest edge pricing
  Crowder and Hattingh, MP Study **4** (1975) 12–25

Kaul's algorithm: approximate steepest edge pricing

- Implementation of Kaul's algorithm uses approximate steepest edge pricing
  Crowder and Hattingh, MP Study **4** (1975) 12–25
  - Contributions to weights from Z_1 are exact
  - Contributions from Z_2 are unit then updated exactly

Kaul's algorithm: approximate steepest edge pricing

- Implementation of Kaul's algorithm uses approximate steepest edge pricing
  Crowder and Hattingh, MP Study **4** (1975) 12–25
  - Contributions to weights from Z_1 are exact
  - Contributions from Z_2 are unit then updated exactly
- Reduces the number of iterations required to solve the problem by a factor of 2–30
- Adds minimal computational cost per iteration $\Rightarrow$ major net saving
- Improves the numerical stability

Kaul's algorithm: results

- Very limited results to report
- On a small problem (2410 rows, 3864 columns)
  - Standard simplex: 30 iterations/second
  - Kaul's algorithm: 70 iterations/second
- Will improve with problem size!

Kaul's algorithm: dense parallel implementation

- The update of each of the dense matrices involves an outer product operation
- Techniques for the parallel standard simplex method can be used
- We will implement this on SunFire E15K and UoE's new 512 processor cluster

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm
- Observe that for

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}E\Lambda \\ -\Lambda R W^{-1} & \Lambda R W^{-1} E \Lambda + \Lambda \end{bmatrix}$$

where $W = D - E\Lambda R$ (and $\Lambda = T^{-1}$)

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm
- Observe that for

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}E\Lambda \\ -\Lambda R W^{-1} & \Lambda R W^{-1} E \Lambda + \Lambda \end{bmatrix}$$

  where $W = D - E\Lambda R$ (and $\Lambda = T^{-1}$)
- Sparse implementation thus requires
  - Dense factorisation of "small" matrix W

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm
- Observe that for

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}E\Lambda \\ -\Lambda R W^{-1} & \Lambda R W^{-1} E \Lambda + \Lambda \end{bmatrix}$$

  where $W = D - E\Lambda R$ (and $\Lambda = T^{-1}$)
- Sparse implementation thus requires
  - Dense factorisation of "small" matrix W
  - Sparse factored (product form) of Λ_i, for $i = 1, \ldots, r$

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm
- Observe that for

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}E\Lambda \\ -\Lambda R W^{-1} & \Lambda R W^{-1} E \Lambda + \Lambda \end{bmatrix}$$

  where $W = D - E\Lambda R$ (and $\Lambda = T^{-1}$)
- Sparse implementation thus requires
  - Dense factorisation of "small" matrix W
  - Sparse factored (product form) of Λ_i, for $i = 1, \ldots, r$
  - Products with submatrices of constraint matrix A

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm
- Observe that for

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}E\Lambda \\ -\Lambda R W^{-1} & \Lambda R W^{-1} E \Lambda + \Lambda \end{bmatrix}$$

  where $W = D - E\Lambda R$ (and $\Lambda = T^{-1}$)
- Sparse implementation thus requires
  - Dense factorisation of "small" matrix W
  - Sparse factored (product form) of Λ_i, for $i = 1, \ldots, r$
  - Products with submatrices of constraint matrix A
- Parallelisation of sparse implementation requires
  - Parallel matrix-vector products
  - Concurrent serial operations with Λ_i, for $i = 1, \ldots, r$

Kaul's algorithm: sparse implementation

- Based on revised simplex view of Kaul's algorithm
- Observe that for

$$B = \begin{bmatrix} D & E \\ R & T \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} W^{-1} & -W^{-1}E\Lambda \\ -\Lambda R W^{-1} & \Lambda R W^{-1} E \Lambda + \Lambda \end{bmatrix}$$

  where $W = D - E\Lambda R$ (and $\Lambda = T^{-1}$)
- Sparse implementation thus requires
  - Dense factorisation of "small" matrix W
  - Sparse factored (product form) of Λ_i, for $i = 1, \ldots, r$
  - Products with submatrices of constraint matrix A
- Parallelisation of sparse implementation requires
  - Parallel matrix-vector products
  - Concurrent serial operations with Λ_i, for $i = 1, \ldots, r$
- Scope for scalable massively parallel implementation

The next steps

- Dense serial implementation is nearing completion

The next steps

- Dense serial implementation is nearing completion
- Dense parallel implementation is planned

The next steps

- Dense serial implementation is nearing completion
- Dense parallel implementation is planned
- Sparse serial implementation is planned

The next steps

- Dense serial implementation is nearing completion
- Dense parallel implementation is planned
- Sparse serial implementation is planned
- Sparse parallel implementation is next logical step

  Should yield a powerful solver for large scale block-angular LP problems

The next steps

- Dense serial implementation is nearing completion

The next steps

- Dense serial implementation is nearing completion
- Dense parallel implementation is planned

The next steps

- Dense serial implementation is nearing completion
- Dense parallel implementation is planned
- Sparse serial implementation is planned

The next steps

- Dense serial implementation is nearing completion
- Dense parallel implementation is planned
- Sparse serial implementation is planned
- Sparse parallel implementation is next logical step

  Should yield a powerful solver for large scale block-angular LP problems