Parallelisation of the revised simplex method for general large scale LP problems

Julian Hall

School of Mathematics

University of Edinburgh

21st June 2004

Overview

- The simplex method for linear programming
  - The standard simplex method
  - The revised simplex method
- Past approaches to parallelising the simplex method
- SYNPLEX
- Results and conclusions

Solving LP problems

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

- At any vertex the variables may be partitioned into index sets
  - $\mathcal{B}$ of m basic variables $\boldsymbol{x}_B \geq \boldsymbol{0}$:
  - $\mathcal{N}$ of $n - m$ nonbasic variables $\boldsymbol{x}_N = \boldsymbol{0}$.
- Components of $\boldsymbol{c}$ and columns of A are
  - the basic costs $\boldsymbol{c}_B$ and basis matrix B;
  - the non-basic costs $\boldsymbol{c}_N$ and matrix N.

Reduced LP problem

At any vertex the original problem is

$$\begin{array}{llcccccl} \text{minimize} & f & = & \boldsymbol{c}_N^T\boldsymbol{x}_N & + & \boldsymbol{c}_B^T\boldsymbol{x}_B & & \\ \text{subject to} & & & N\,\boldsymbol{x}_N & + & B\,\boldsymbol{x}_B & = & \boldsymbol{b} \\ & & & \multicolumn{5}{c}{\boldsymbol{x}_N \geq \mathbf{0} \quad \boldsymbol{x}_B \geq \mathbf{0}.} \end{array}$$

Eliminate $\boldsymbol{x}_B$ from the objective to give

$$\begin{array}{llcccccl} \text{minimize} & f & = & \hat{\boldsymbol{c}}_N^T\boldsymbol{x}_N & + & \hat{f} & & \\ \text{subject to} & & & \hat{N}\,\boldsymbol{x}_N & + & I\,\boldsymbol{x}_B & = & \hat{\boldsymbol{b}} \\ & & & \multicolumn{5}{c}{\boldsymbol{x}_N \geq \mathbf{0} \quad \boldsymbol{x}_B \geq \mathbf{0},} \end{array}$$

where $\hat{\boldsymbol{b}} = B^{-1}\boldsymbol{b}$, $\hat{N} = B^{-1}N$ and $\hat{\boldsymbol{c}}_N$ is the vector of **reduced costs** given by

$$\hat{\boldsymbol{c}}_N^T = \boldsymbol{c}_N^T - \boldsymbol{c}_B^T\hat{N}.$$

The standard simplex method

	$\mathcal{N}$	$\mathcal{B}$	RHS
1 $\vdots$ m	$\hat{N}$	I	$\hat{\boldsymbol{b}}$
0	$\hat{\boldsymbol{c}}_N^T$	$\boldsymbol{0}^T$	$-\hat{f}$

In each iteration:

- Select the column q' of a nonbasic variable $q \in \mathcal{N}$ to be increased from zero.
- Find the row p of the first basic variable $p' \in \mathcal{B}$ to be zeroed.
- Exchange indices p' and q between sets $\mathcal{B}$ and $\mathcal{N}$.
- Update tableau corresponding to this **basis change**.
  - Scale row p by $\hat{N}_{pq'}$.
  - Eliminate $\hat{N}_{iq'}$, $i = 0, \ldots, m, i \neq p$.

Revised simplex method

Given $\hat{\boldsymbol{c}}_N$, $\hat{\boldsymbol{b}}$ and a representation of B^{-1}, repeat

`CHUZC:` Scan the reduced costs $\hat{\boldsymbol{c}}_N$ for a good candidate q to enter the basis.
`FTRAN:` Form the pivotal column $\hat{\boldsymbol{a}}_q = B^{-1}\boldsymbol{a}_q$, where $\boldsymbol{a}_q$ is column q of A.
`CHUZR:` Scan the ratios $\hat{b}_i/\hat{a}_{iq}$ for the row p of a good candidate to leave the basis. Let $\alpha = \hat{b}_p/\hat{a}_{pq}$.
Update $\hat{\boldsymbol{b}} := \hat{\boldsymbol{b}} - \alpha\hat{\boldsymbol{a}}_q$.
`BTRAN:` Form $\boldsymbol{\pi}_p^T = \boldsymbol{e}_p^T B^{-1}$.
`PRICE:` Form the pivotal row $\hat{\boldsymbol{a}}_p^T = \boldsymbol{\pi}_p^T N$.
Update reduced costs $\hat{\boldsymbol{c}}_N^T := \hat{\boldsymbol{c}}_N^T - \hat{c}_q\hat{\boldsymbol{a}}_p^T$.
`If` (growth in factors) `then`
  `INVERT:` Form a representation of B^{-1}.
`else`
  `UPDATE:` Update the representation of B^{-1} corresponding to the basis change.
`end if`

Factored representation of B^{-1}

When using the product form update

$$B^{-1} = E_U^{-1} B_0^{-1}$$

- B_0^{-1} is represented by the INVERT etas
- E_U^{-1} is represented by the UPDATE etas
- FTRAN is performed as

$$\tilde{\boldsymbol{a}}_q = B_0^{-1} \boldsymbol{a}_q \quad \text{and} \quad \hat{\boldsymbol{a}}_q = E_U^{-1} \tilde{\boldsymbol{a}}_q.$$

- BTRAN is performed as

$$\tilde{\boldsymbol{\pi}}_p{}^T = \boldsymbol{e}_p^T E_U^{-1} \quad \text{and} \quad \boldsymbol{\pi}_p^T = \tilde{\boldsymbol{\pi}}_p{}^T B_0^{-1}.$$

Parallelising the simplex method

- Why try to parallelise the simplex method?
  - Want to solve general large sparse LP problems fast
  - Serial revised simplex solvers are very efficient
    CPLEX, OSL, Xpress, SOPLEX, EMSOL
  - Parallel solver must offer (hope of) further performance improvement
- How to parallelise the simplex method
  - Exploit **data parallelism**
    Use several processors simultaneously to perform a single operation
  - Exploit **functional parallelism**
    Perform more than one operation simultaneously using several processors
- Architectures
  - Distributed memory machines
    Communication between processors using message-passing routines (PVM/MPI)
  - Shared memory machines
    Coordination of processes and memory read/write using message-passing routines or compiler directives (OpenMP)

Standard simplex method

- Tableau update yields simple data parallelism using dense data structure
- Worthwhile exercise in (early days of) parallel computing

  Finkel (1987), Stunkel (1988), Agrawal *et al.* (1989), Luo *et al.* (1988–91)
- Notable implementations due to
  - Eckstein *et al.* (1995): Massively parallel (CM-5)
  - Lentini *et al.* (1995): Parallel *sparse* standard simplex
- Parallel efficiency of 50%–90%
- Totally uncompetitive with serial RSM unless a prohibitively large number of processors are used
- Still being considered

  Yarmish (2001), Elangovan (2004)

Revised simplex method

- Immediate data parallelism only in PRICE
- Limited data parallelism in FTRAN, BTRAN and INVERT
- Theoretical discussion by
  - Pfefferkorn and Tomlin (1976): for ILLIAC IV
  - Helgason *et al.* (1988): using quadrant interlocking factorization
  - McKinnon and Plab (1997): processing of factored inverse and parallel FTRAN
- Implementation by
  - Ho and Sundarraj (1994)
    Data-parallel PRICE and functionally-parallel INVERT
  - Shu (1995)

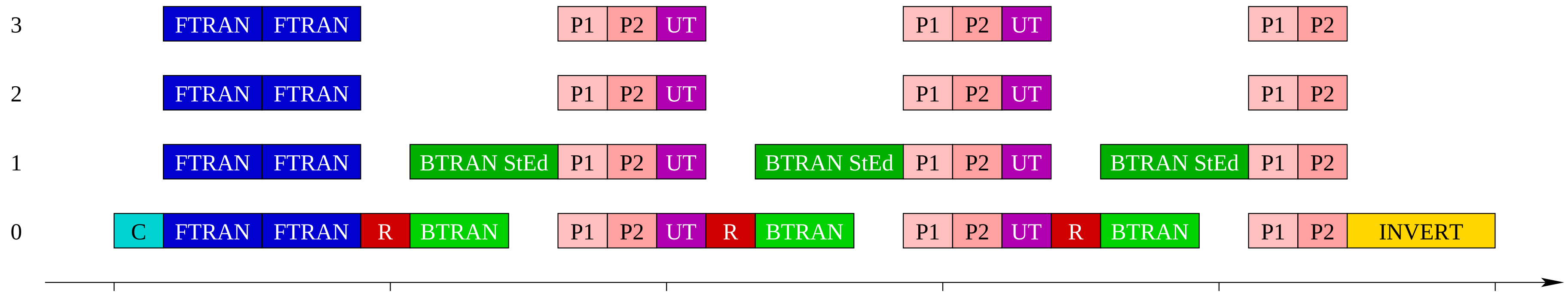
  Significant speed-up only obtained when $n \gg m$ so PRICE dominates

Dual (revised) simplex method

- Bixby and Martin (1995–2000)
- Dual simplex can require far fewer iterations than primal simplex
- Requires full PRICE
- Only exploited data parallelism in PRICE
- Good results on shared memory machine relative to serial CPLEX for problems when $n \gg m$

RSM with steepest edge

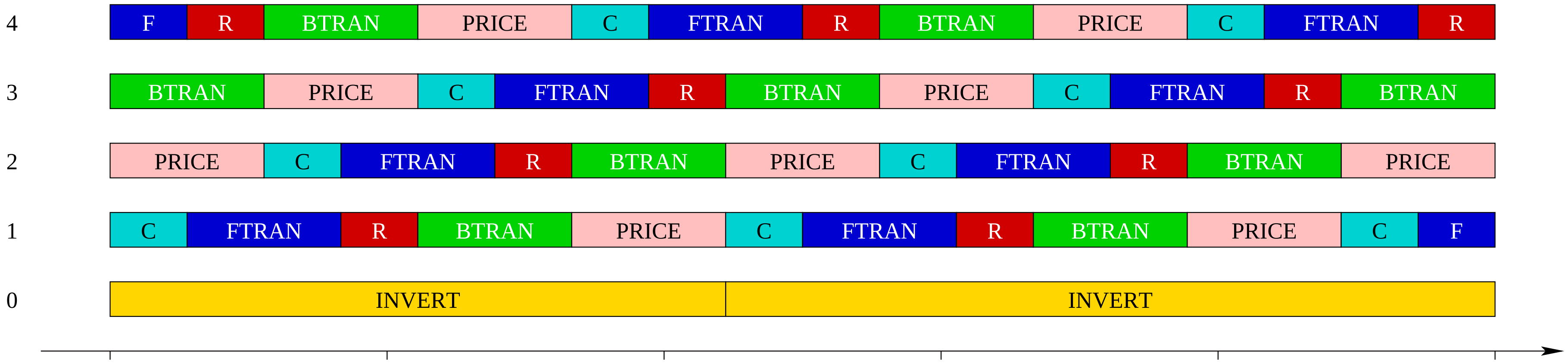
- Wunderling (1996)
- Steepest edge pricing requires additional BTRAN and PRICE each iteration
  Number of iterations to solve an LP can be significantly reduced
- Functional parallelism plus parallel PRICE
  - Multiple FTRAN computes a number of attractive tableau columns
  - Two BTRANs performed simultaneously
  - Functionally parallel INVERT?



- Fully parallel for only two processors
- Good results only for problems when $n \gg m$

ASYNPLEX

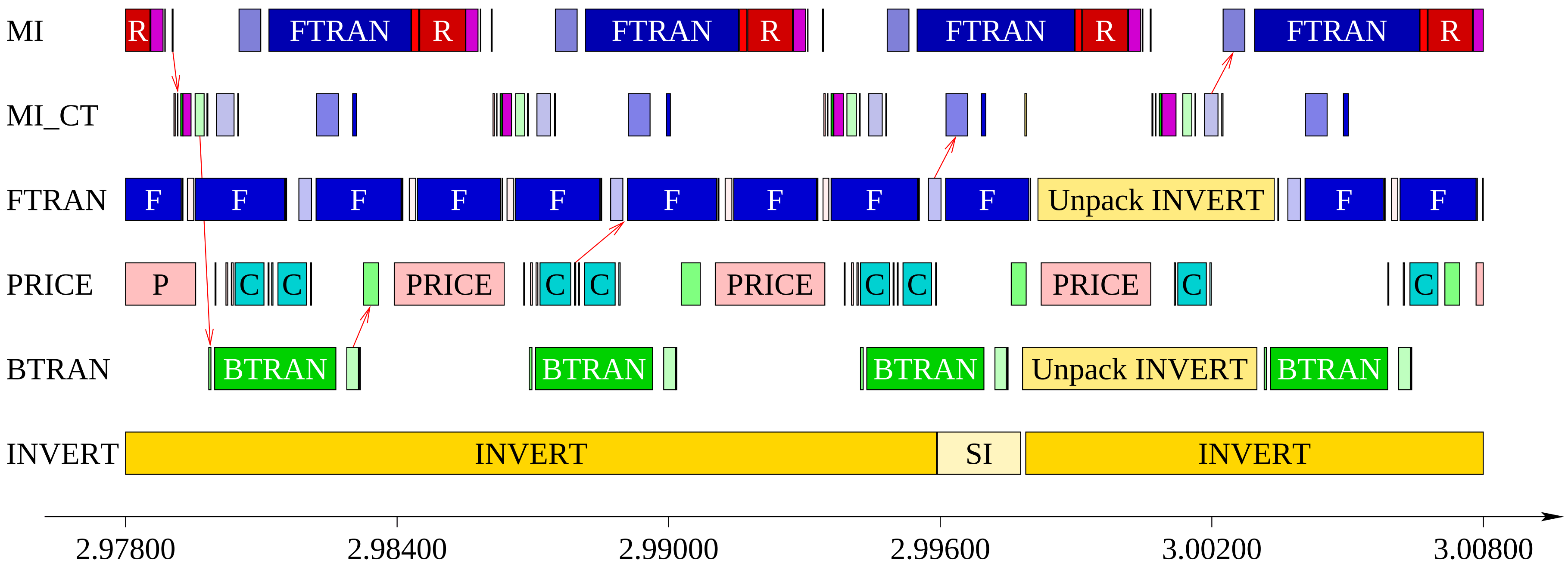
- Hall and McKinnon (1995)
- Dantzig pricing with 'full' BTRAN and PRICE
- Functional parallelism
- Target platform: Cray T3D (PVM and **`shmem`**)



- Speed-up of up to 5 on modest Netlib problems
- Performance limited by
  - communication overhead
  - duplicated work
  - limited candidate persistence
- Numerically unstable

PARSMI

- Hall and McKinnon (1996)
- SSM minor iterations on small subset of tableau columns
- Each processor dedicated to a particular operation
- Devex pricing requires no more operations per iteration
  Number of iterations to solve an LP can be significantly reduced



SYNPLEX

- Synchronous variant of PARSMI
- Better candidate persistency
- INVERT not overlapped with basis changes so numerically stable
- All tasks distributed over all processors
- Target platform: Sun Fire E15k (52 x 0.9GHz UltraSPARC III)
- Parallelise using OpenMP

SYNPLEX: parallel scheme

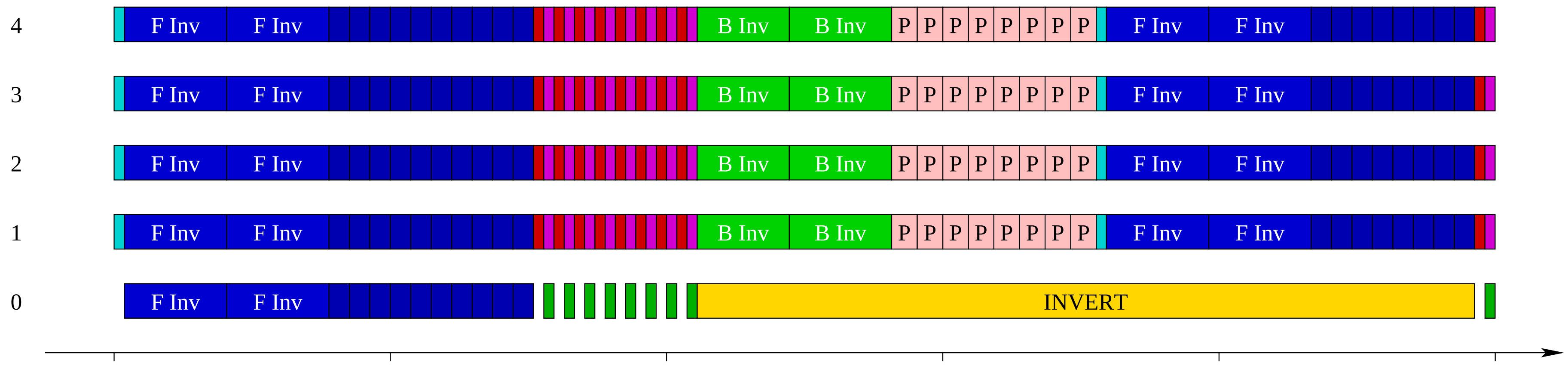
When using $1 + p$ processors

- Functionally parallel multiple FTRAN for INVERT etas
- Rows distributed over p processors for data parallel
  - multiple FTRAN for UPDATE etas
  - CHUZR
  - UPDATE RHS
- BTRAN for UPDATE etas performed on 1 processor
  Overlapped with UPDATE RHS

SYNPLEX: parallel scheme (cont.)

- Functionally parallel multiple BTRAN for INVERT etas
- Columns distributed over p processors for data parallel
  - PRICE
  - CHUZC
- INVERT overlapped with
  - BTRAN for INVERT etas
  - PRICE
  - CHUZC
  - FTRAN

SYNPLEX in theory



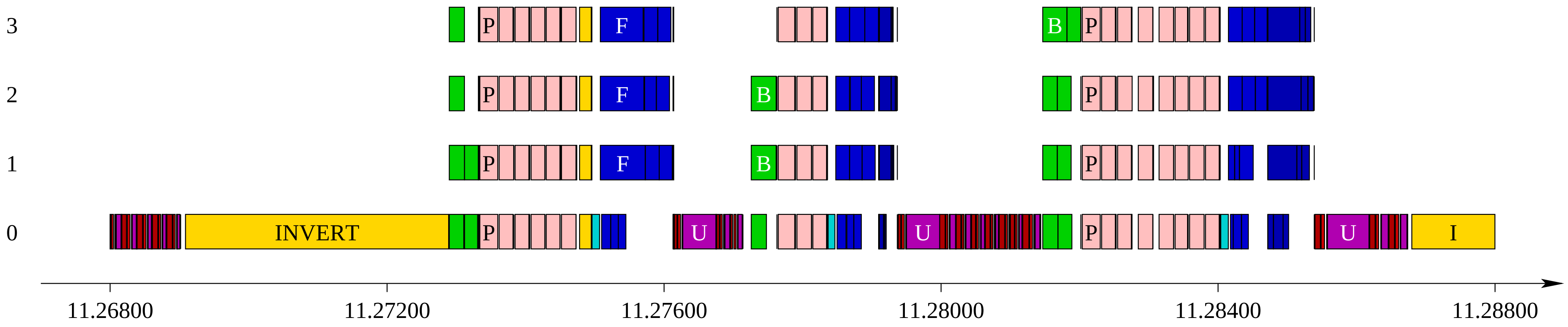
Implementation

- SYNPLEX not yet fully implemented
  - Phase 2
  - Dantzig pricing
- SYNPLEX not yet efficient in serial
  - Column-wise PRICE
  - Doesn't exploit 'hypersparsity'
- SYNPLEX not yet fully parallelised
  - INVERT not overlapped
  - Minor iterations still sequential
  - Inefficient parallel UPDATE FTRAN

Results: 25FV47

16264	FTRANs
5695	Iterations
35%	Candidate persistence

PE	Solve	FTRAN	BTRAN	PRICE
	CPU(s)			
1	8.12	2.76	1.09	1.53
	Speed-up			
2	0.96	1.89	1.53	0.99
3	1.05	2.65	1.95	1.18
4	1.08	3.41	2.27	1.26

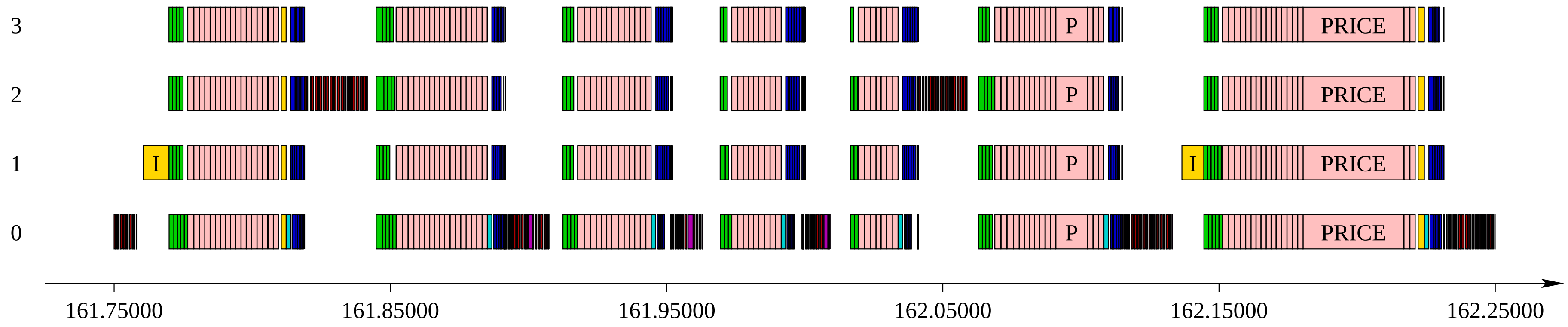


Description	Elapsed	Description	Elapsed
PRICE	0.01441	Update Tableau	0.00250
FTRAN: Inv Etas	0.00721	Pack pivotal column	0.00122
INVERT	0.00500	Initialise UPDATE eta file	0.00068
BTRAN: Inv Etas	0.00399	CHUZR	0.00067
FTRAN: U Etas	0.00294	All threads barrier	0.00043

Results: PDS-06

88685	FTRANs
28989	Iterations
32%	Candidate persistence

PE	Solve	FTRAN	BTRAN	PRICE
	CPU(s)			
1	187.69	49.91	31.49	66.02
	Speed-up			
2	1.02	1.84	1.55	1.05
3	1.12	2.58	2.02	1.25
4	1.17	3.32	2.42	1.41



Description	Elapsed	Description	Elapsed
PRICE	0.92479	Initialise UPDATE eta file	0.01596
BTRAN: Inv Etas	0.12884	Major CHUZC	0.01151
FTRAN: Inv Etas	0.12115	FTRAN: U Etas	0.00794
Pack pivotal column	0.05126	Update Tableau	0.00762
INVERT	0.01717	CHUZR	0.00455

Conclusions

- Shared memory machines with OpenMP offer prospect of
  - easier implementation
  - reduced parallel overheads
  - worthwhile target platform
- SYNPLEX addresses problems of candidate persistency and numerical stability
- Crude and incomplete implementation already offers some speed-up
- Potential for worthwhile speed-up