Performance of linear solvers in Interior Point Methods

Jonathan Hogg
Jennifer Scott

Scientific Computing Department
STFC Rutherford Appleton Laboratory

Software

Results in this paper use:

IPOPT[1]

Interior point optimization code for nonlinear problems
Filter line-search approach
Written by Wächter and Biegler
Part of COIN-OR project
Open Source Eclipse Licence

Probably most widely used open-source IPM solver
Interfaces to a number of linear solvers

However our solvers are also used in a number of other codes.

[1] A. Wächter and L. T. Biegler, *On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming*, Mathematical Programming 106(1), pp. 25-57, 2006

What's happening

Very simplistically...

while not converged **do**
  Find a descent direction.
  Conduct a line search for next trial point.
    (Repeat with second-order corrections).
  Take the step; update parameters.
end while

Find a descent direction

Solve

$$Ax = b$$

where

$$A = \begin{pmatrix} W + \Sigma_k + \delta_w I & J \\ J^T & -\delta_c I \end{pmatrix}$$

By sparse direct method i.e. factorize with pivoting

$$A = LDL^T$$

Science & Technology Facilities Council

Requirements and options

- Get the “right” answer. Inaccuracy $\Rightarrow$ more IPM iterations.
- Report correct inertia — required for filter line search to work

Two main options:

1. Static pivoting — if a pivot is too small, add something to it.
   Faster, less accurate
2. Threshold pivoting — if a pivot is small, delay until later.
   Slower, more accurate

Science & Technology Facilities Council

HSL_MA97

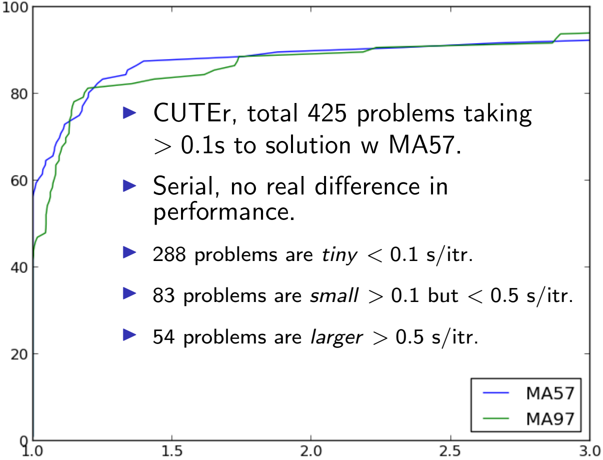
Recently developed a new multicore code (OpenMP)
Designed for bit-compatibility and all problem sizes.
Handles both positive-definite and indefinite systems.

Problem	n	MA57	HSL_MA97		
		1	1	8	Speedup
GHS_indef/a0nsdsil	80016	0.054	0.055	0.055	1.00
Boeing/bcsstk39	46772	0.63	0.55	0.314	1.74
Oberwolfach/t3dh	79171	13.3	10.6	2.57	4.13
ND/nd12k	36000	109	101	19.7	5.11
Oberwolfach/bone010	986703	682	553	84.4	6.55

(MA57 is popular choice for IPOPT; it is also used in MATLAB)

So just using it in IPOPT should work well, right?

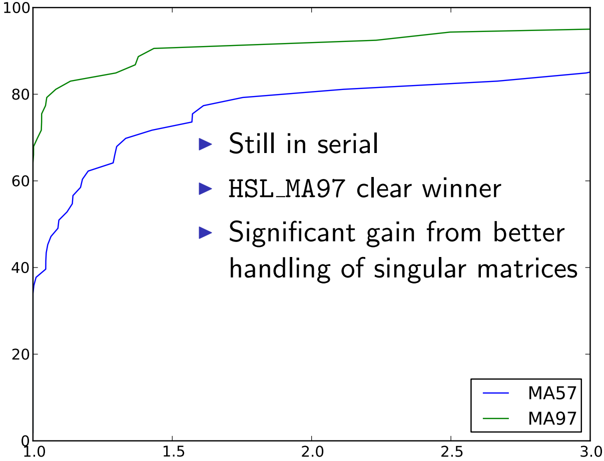
MA57 vs HSL_MA97



- CUTEr, total 425 problems taking > 0.1s to solution w MA57.
- Serial, no real difference in performance.
- 288 problems are *tiny* < 0.1 s/itr.
- 83 problems are *small* > 0.1 but < 0.5 s/itr.
- 54 problems are *larger* > 0.5 s/itr.

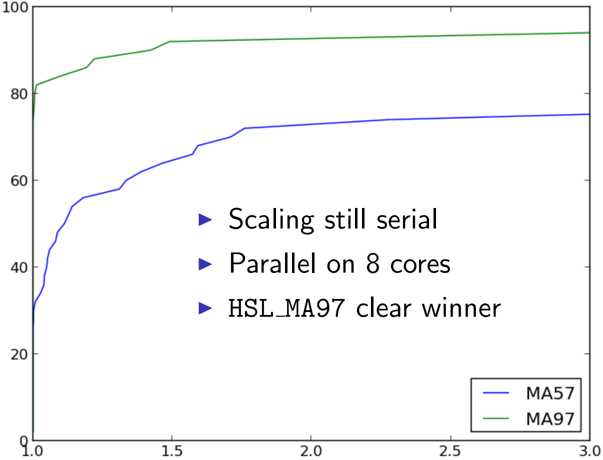
Science & Technology Facilities Council

MA57 vs HSL_MA97, Larger problems only



- Still in serial
- HSL_MA97 clear winner
- Significant gain from better handling of singular matrices

Parallel MA57 vs HSL_MA97, Larger problems only



- Scaling still serial
- Parallel on 8 cores
- HSL_MA97 clear winner

Comparison pitfalls

Hard to compare across solvers/scalings:

- Different paths to optimum
- Different optima
- Different matrices

Comparison pitfalls

Hard to compare across solvers/scalings:

- Different paths to optimum
- Different optima
- Different matrices
- Small differences in detected inertia invoke different code

The need for scaling

- Reduces pivoting required (fewer delayed pivots)
- Increases predictability
- Increases accuracy

The need for scaling

- Reduces pivoting required (fewer delayed pivots)
- Increases predictability
- Increases accuracy

$\Rightarrow$ Increases Parallelism

The need for scaling

- Reduces pivoting required (fewer delayed pivots)
- Increases predictability
- Increases accuracy

$\Rightarrow$ Increases Parallelism*

* (But not for many problems.)

Science & Technology Facilities Council

Trying everything

Good scalings speed up factorizations of poorly scaled matrices.

None Free?
Fastest; can cause many delayed pivots.

MC19 Very cheap.
Faster; can cause many delayed pivots.

MC64 Find weighted maximum matching.
Good; can be slow.

MC77 Several matrix-vector multiplies.
Fast; can be insufficient.

MC80 MC64 + reordering.
Very good; can be very slow.

Best approach varies by problem.

Trying everything

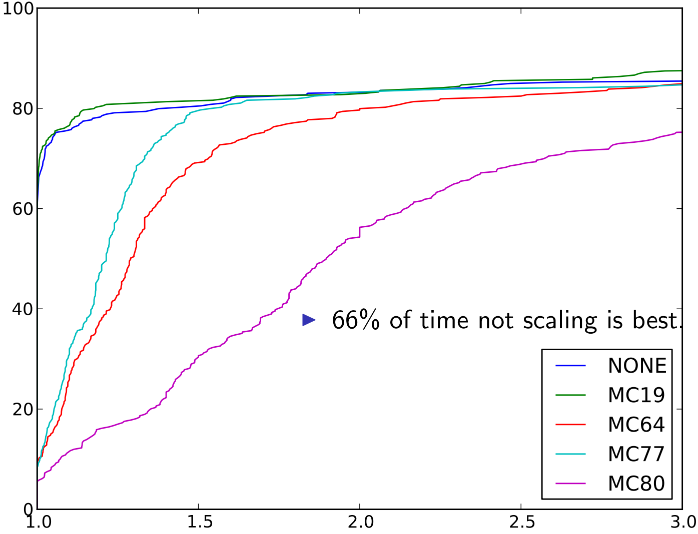
Good scalings speed up factorizations of poorly scaled matrices.

None	Free? Fastest; can cause many delayed pivots.	Trivial
MC19	Very cheap. Faster; can cause many delayed pivots.	?
MC64	Find weighted maximum matching. Good; can be slow.	Not parallel
MC77	Several matrix-vector multiplies. Fast; can be insufficient.	Parallelisable
MC80	MC64 + reordering. Very good; can be very slow.	Not parallel

Best approach varies by problem.

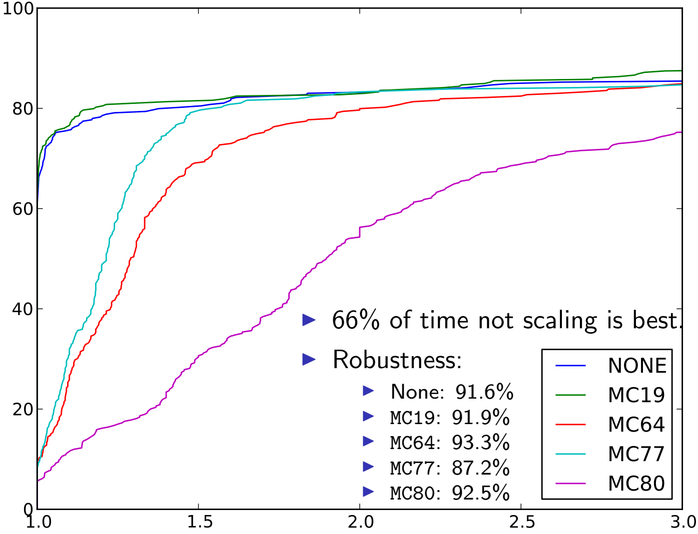
Science & Technology Facilities Council

Trying everything — results



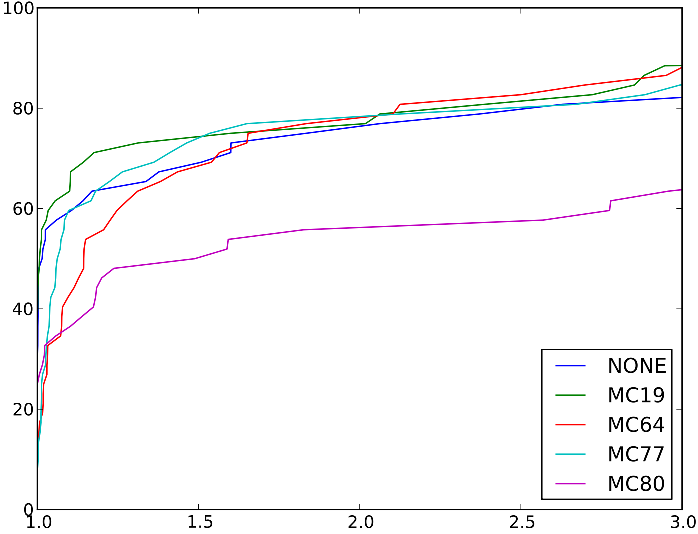
- 66% of time not scaling is best.

Trying everything — results



- 66% of time not scaling is best.
- Robustness:
  - None: 91.6%
  - `MC19`: 91.9%
  - `MC64`: 93.3%
  - `MC77`: 87.2%
  - `MC80`: 92.5%

Trying everything — results, larger only



Scaling examples

Times

Problem	None	MC64	MC77	MC80
CBRATU3D	1.56	1.52	1.52	**0.28**
EIGENA	138.3	92.9	**86.4**	98.4
ELEC	88.1	**57.3**	60.6	85.3
HADAMALS	**15.4**	19.8	17.6	27.4

Iterations

Problem	None	MC64	MC77	MC80
CBRATU3D	3	3	3	3
EIGENA	34	34	34	34
ELEC	342	191	216	206
HADAMALS	307	306	306	306

Explanations

- Different paths (Inertia!)
- More work (Delayed Pivots!)

Inertia and scalings...

Based on problem A2ENSNDL

$$\begin{pmatrix} 1e\text{-}22 & & & & & \\ & 1e\text{-}22 & & & & \\ & & 1e\text{-}22 & & & \\ & & & \ddots & & \\ 0.3 & 0.2 & 0.4 & \dots & 1e10 & \\ 0.2 & 0.3 & 0.5 & \dots & 0.5 & 1e10 \end{pmatrix}$$

What is the inertia?

No scaling: Inertia (2,2,996), Maximum front 1000×1000

Inertia and scalings...

Based on problem A2ENSNDL

$$\begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & \ddots & & \\ 3e\text{-}11 & 2e\text{-}11 & 4e\text{-}11 & \dots & 1 & \\ 2e\text{-}11 & 3e\text{-}11 & 5e\text{-}11 & \dots & 5e\text{-}11 & 1 \end{pmatrix}$$

What is the inertia?

No scaling: Inertia (2,2,996), Maximum front 1000×1000

MC64: Inertia (500,500,0), Maximum front 2×2

Inertia and scalings...

Based on problem A2ENSNDL

$$\begin{pmatrix} 1e\text{-}19 & & & & & \\ & 1e\text{-}19 & & & & \\ & & 1e\text{-}19 & & & \\ & & & \ddots & & \\ 3e\text{-}11 & 2e\text{-}11 & 4e\text{-}11 & \dots & 1 & \\ 2e\text{-}11 & 3e\text{-}11 & 5e\text{-}11 & \dots & 5e\text{-}11 & 1 \end{pmatrix}$$

What is the inertia?

No scaling: Inertia (2,2,996), Maximum front 1000×1000

MC64: Inertia (500,500,0), Maximum front 2×2

Scaling 3: Inertia (500,500,0), Maximum front 1000×1000

Parallelism

Still a work in progress:

- Maximum speedup at present 2.35
- Most speedups on large problem in range 1.40–1.80.

Because:

- Good scalings are still very serial
  (and can be 70% of run time).
- Not using a good scaling limits parallelism.
- Problems tested very small by direct methods standards.

Open questions?

- Can we get a high quality parallel scaling?
- How should inertia detection be handled with respect to scaling? What is zero?
- Better weak scaling: better speedup on small matrices.
- Can we get better parallel performance by driving parallelism up into the IPM somehow?

General HSL: http://www.hsl.rl.ac.uk

HSL IPOPT: http://www.hsl.rl.ac.uk/ipopt

HSL is freely available to academics

Questions?

General HSL: http://www.hsl.rl.ac.uk
HSL IPOPT: http://www.hsl.rl.ac.uk/ipopt

HSL is freely available to academics