HPC Solution Methods for Large Sparse Linear Equation Systems

Rashid Mehmood
School of Computing and Engineering
University of Huddersfield
R.Mehmood@hud.ac.uk

16 January 2013
The Cockcroft Institute, Daresbury Laboratory, Warrington

Outline

- Context
- Solution Methods and Results
  - Sparse linear equations systems with over a billion equations/unknowns
- Conclusions and thoughts

Context

- Markov Chains and Queueing Theory
  - Markov Decision Processes (MDPs) and Discrete State models
  - widely used analysis, optimization and decision making tools
  - in many areas of science and engineering
- Real life systems could be modelled and analysed
  - for their steady-state and transient behaviour
  - Performance measures e.g. no service probability of a system
  - can be calculated by computing the probability distributions

…Context

- A major hurdle in the applicability of these tools to complex and large problems
  - the curse of dimensionality problem
  - models for even trivial real life systems comprise millions of states
  - hence require intelligent high performance computing solutions

- this talk
  - briefly report on our experiences and developed techniques to address the curse of dimensionality

The Numerical Problem

- Final Phase: Transient and Steady State Solution
  - Differential equation for transient
    - $\partial x(t) / \partial t = x(t)A$
  - Steady-state solution
    - $Ax = 0, \sum x_i = 1, x = \lim_{t \to \infty} x(t)$
  - Optimisation Problems (MDPs)
    - $Ax \geq 0$ or $Ax \leq 0$
- Common Element: Matrix Computations

...The Numerical Problem

- Challenge: State-space explosion
  - numerical solution of Ax = 0
  - Matrix computations involving large matrices and vectors
  - The matrix A is sparse

State-Space Explosion

- Solution of $\mathrm{Ax} = 0$
  - requires storage of the matrix A and the vector(s) x
- Numerical methods: need fast but low in storage
  - Direct and iterative methods
    - Gaussian elimination (fill-in)
    - Jacobi, Power, Gauss-Seidel (slow, low in storage)
    - Krylov subspace methods (fast, high in storage)

...State-Space Explosion

- Storage: need compact storage but fast access
  - Implicit methods: BDD-based storage
  - Explicit methods: methods from linear algebra community
  - Parallel and out-of-core techniques
  - Seek storage and CPU alternatives

Markov Process

- A Markov Process is
  - a Stochastic Process: $\{x(t) \mid t \in T\}$
  - for any $t_0 < t_1 < t_2 < \ldots < t_k < t$
  - the conditional distribution of $X(t)$ depends only on $x(t_k)$
  - (the values taken by $x(t)$ are process states)

- Intuitively…

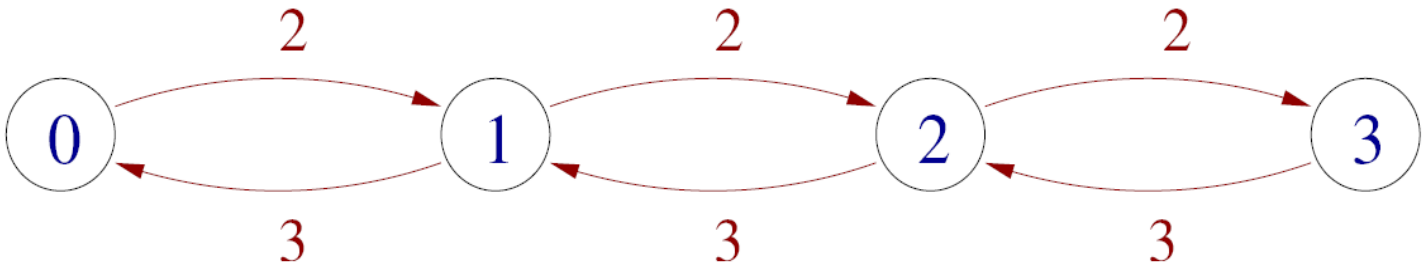
- Mathematically:
  - $P[x(t) \leq x \mid x(t_k) = x_k, x(t_{k-1}) = x_{k-1}, \ldots, x(t_0) = x_0]$
  - equals $P[x(t) \leq x \mid x(t_k) = x_k]$

Markov Chain

- Continuous-time and discrete-time
- A CTMC:
  - a set of states, S
  - a transition rate matrix, $R: S \times S \longrightarrow R$
  - state-to-state transition: if $r_{i,j} > 0$
  - the mean sojourn time for state i is $1/E(i)$
  - $E(i) = \sum_{j \epsilon S,\, j \neq i} r_{i,j}$
  - Transition probability: state i to j within t time units: $1 - e^{-E(i)t}$
  - $q_{i,i} = - E(i)$
  - CTMC generator sparse matrix Q

Transition Diagram and Matrix

- A Simple Example
  - Markov Chain: Web Server or a Road Network



$$\begin{pmatrix} -2 & 2 & 0 & 0 \\ 3 & -5 & 2 & 0 \\ 0 & 3 & -5 & 2 \\ 0 & 0 & 3 & -3 \end{pmatrix}$$

Steady State Solution

- Numerical Steady State Solution
- Probabilities of system to be in a particular state
  - in the long run

$$-2\pi_0 + 3\pi_1 = 0,$$

$$2\pi_0 - 5\pi_1 + 3\pi_2 = 0,$$

$$2\pi_1 - 5\pi_2 + 3\pi_3 = 0,$$

$$2\pi_2 - 3\pi_3 = 0,$$

$$\pi_0 + \pi_1 + \pi_2 + \pi_3 = 1,$$

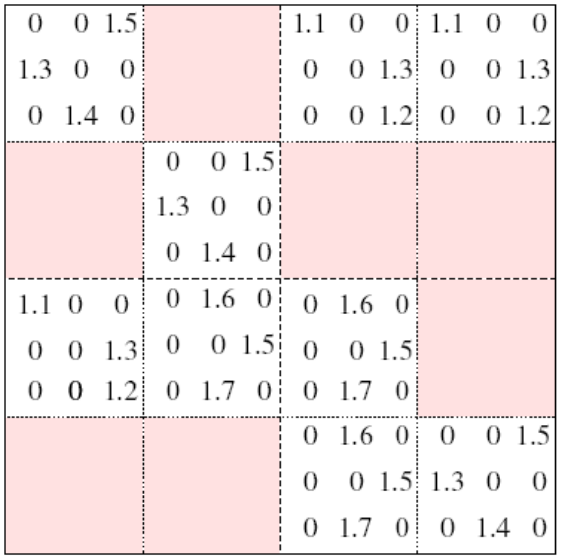
$$\pi = [\frac{27}{65}, \frac{18}{65}, \frac{12}{65}, \frac{8}{65}]$$

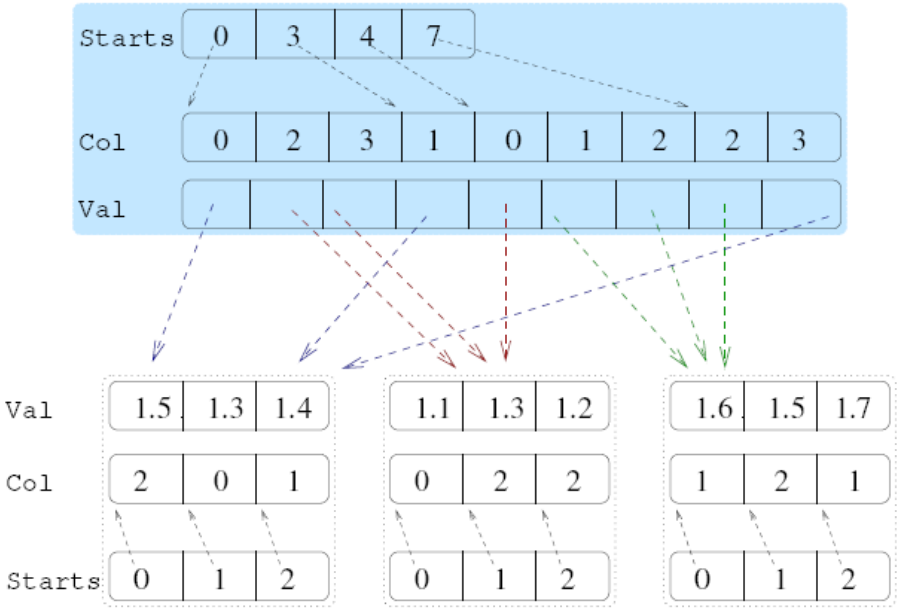
Range of Computational Techniques

- Compact data structures which exploit matrix structure to minimize the memory and computational requirements
- Out-of-core techniques which use concurrent programming (multithreading) and disk storage to efficiently store and retrieve large models
- Parallel computing techniques to use memory and computational power of multiple machines.

The Symbolic CTMC Representation

- Modified Multi-Terminal Binary Decision Diagrams (MTBDDs) [Meh04]
- Decompose matrix into blocks
- Store blocks individually
- Store the high-level information about each block
- Some more compaction



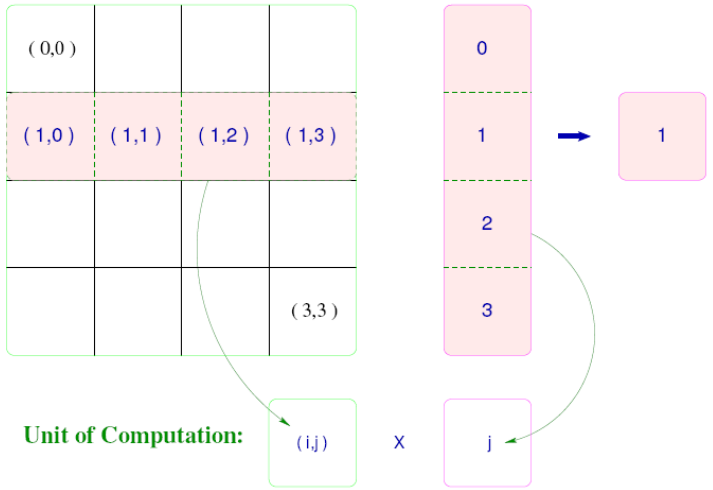


Storage Requirements

k	States (*n*)	a/n	Memory for Matrix (MB)				Vector (MB)
			MSR	Ind. MSR	Comp. MSR	MTBDDs	
FMS Model							
6	537,768	7.82	50	24	17	4	4
10	25,397,658	9.23	2,780	1,366	918	137	194
13	216,427,680	9.87	25,272	12,429	8,354	921	1,651
14	403,259,040	12.35	58,540	28,882	19,382	1,579	3,077
15	724,284,864	12.61	107,297	52,952	35,531	2,676	5,526
Kanban System							
4	454,475	8.76	47	23	16	1	3.5
6	11,261,376	10.27	1,367	674	452	6	86
9	384,392,800	11.64	52,673	25,881	17,435	99	2,933
10	1,005,927,208	11.96	141,535	69,858	46,854	199	7,765
Polling System							
15	737,280	8.3	73	35	24	1	6
21	66,060,288	11.3	8,820	4,334	2,910	66	504
24	603,979,776	12.8	91,008	44,813	30,067	144	1,136
25	1,258,291,200	13.3	196,800	96,960	65,040	317	1,190

An MVP-based Computation

- Partitioning
  - A: $B \times B$ square blocks (not necessary)
  - x: B blocks with n/B entries each
- A unit of Computation: A sub-MVP
  - matrix block $\times$ vector block $(A_{ij} \times X_j)$



Serial Block Jacobi Algorithm

ser_block_Jac($\check{A},\ d,\ b,\ x,\ P,\ n[\,],\ \varepsilon$) {

1. var $\tilde{x},\ Y,\ k \leftarrow 0,$ error $\leftarrow 1.0,\ i,\ j,\ p$
2. while(error $> \varepsilon$)
3. $\quad k \leftarrow k + 1$
4. $\quad$ for($0 \leq i < P$)
5. $\quad\quad Y \leftarrow B_i$
6. $\quad\quad$ for($0 \leq j < P$)
7. $\quad\quad\quad Y \leftarrow Y - \check{A}_{ij} X_j^{(k-1)}$
8. $\quad\quad$ for($0 \leq p < n[i]$)
9. $\quad\quad\quad X_i^{(k)}{}_{[p]} \leftarrow D_i{}_{[p]}{}^{-1} Y_{[p]}$
10. $\quad$ compute error
11. $\quad X_i^{(k-1)} \leftarrow X_i^{(k)}$ }

Out-of-Core Algorithm

Integer constant: B (*number of blocks*)
Semaphores: S_1, S_2: occupied
Shared variables: R_0, R_1 (*To read matrix A blocks into RAM*)

Disk-IO process

```
1.  Local variables: i, j, t = 0
2.  while not converged
3.    for i ← 0 to B − 1
4.      for j ← 0 to B − 1
5.        if not an empty block
6.          disk_read (A_ij, R_t)
7.        Signal(S_1)
8.        Wait(S_2)
9.        t = (t + 1) mod 2
```

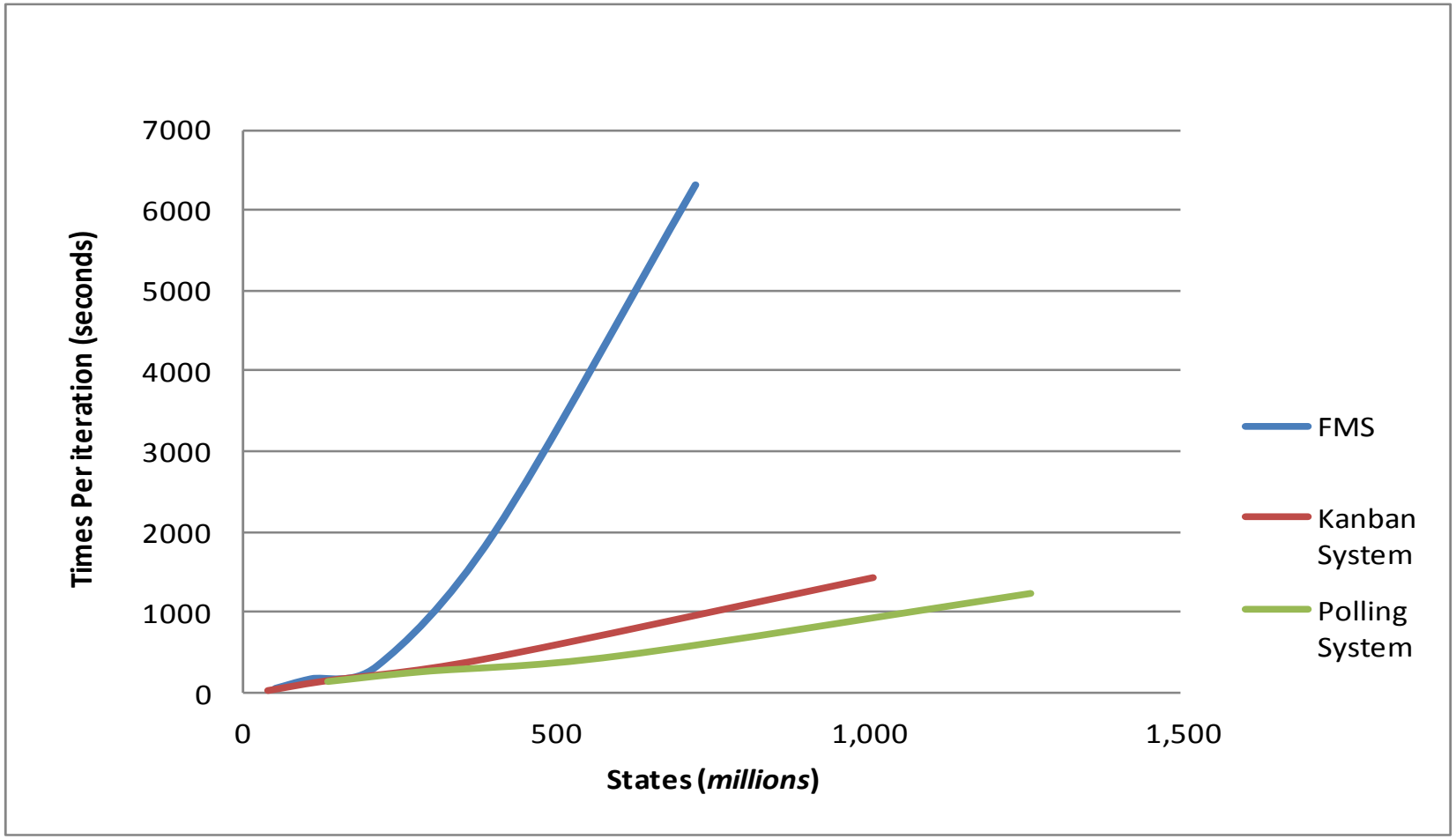
Compute process

```
Local variables: i, j, t = 0
while not converged
 for i ← 0 to B − 1
   for j ← 0 to B − 1
     Wait(S_1)
     Signal(S_2)
     if j ≠ B − 1
       if not an empty block
         sub-MVP(A_ij X_j, R_t)
     else
       Update X_i
       check for convergence
     t = (t + 1) mod 2
```

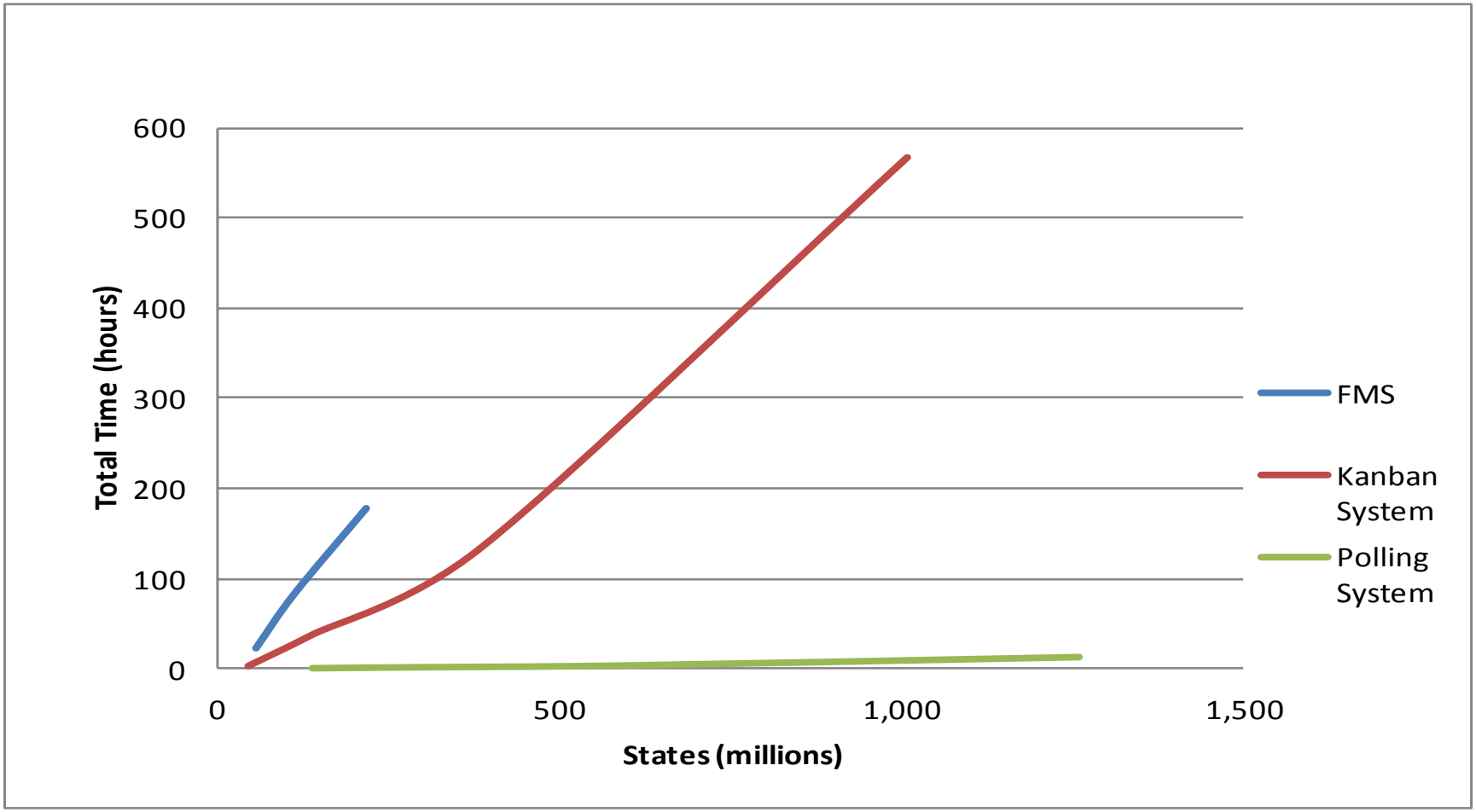
Out-of-Core (Gauss-Seidel) on Single Machine

Model	*k*	States (*n*)	*a* / *n*	Times		Iterations
				Per iteration (seconds)	Total (hr:min:sec)	
FMS	11	54,682,992	9.5	51.6	23:16:39	1624
	12	111,414,940	9.7	170	84:54:20	1798
	13	216,427,680	9.9	327	179:34:39	1977
	14	403,259,040	10.03	1984	-	>50
	15	724,284,864	10.18	6312	-	>50
Kanban System	7	41,644,800	10.8	18.9	4:12:38	802
	8	133,865,325	11.3	139	38:34:21	999
	9	384,392,800	11.6	407	136:54:37	1211
	10	1,005,927,208	11.97	1424	566:49:52	1433
Polling System	22	138,412,032	11.8	143	1:28:11	37
	23	289,406,976	12.3	264	2:47:12	38
	24	603,979,776	12.8	460	4:51:20	38
	25	1,258,291,200	13.3	1226	13:16:54	39

Time Per Iteration



Total Time


600
500
400
300
200
100
0
Total Time (hours)
0
500
1,000
1,500
States (millions)
FMS
Kanban System
Polling System

A Parallel Jacobi Algorithm for p

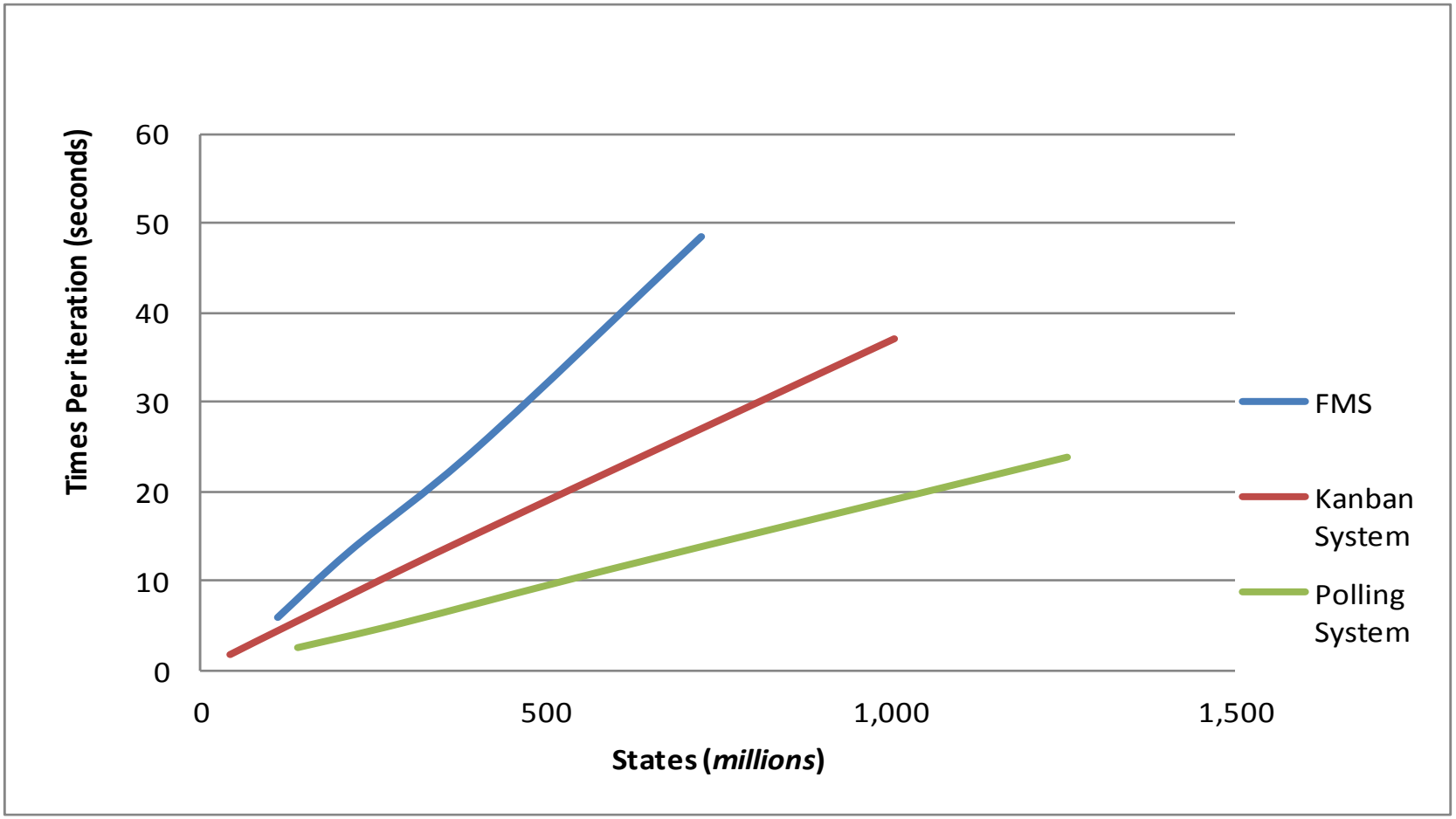
par_block_Jac($\check{A}_p,\ D_p,\ B_p,\ X_p,\ T, N_p,\ \varepsilon$) {

1. var $\tilde{X}_p,\ Z,\ k \leftarrow 0,$ error $\leftarrow 1.0,\ q,\ h,\ i$
2. while(error $> \varepsilon$)
3. $\quad k \leftarrow k + 1;\ \ h \leftarrow 0$
4. $\quad$ for($0 \leq q < T; q \neq p$)
5. $\quad\quad$ if($\check{A}_{pq} \neq \mathbf{0}$)
6. $\quad\quad\quad$ send(request$_{X_q}, q$); $h \leftarrow h + 1$
7. $\quad Z \leftarrow B_p - \check{A}_{pp} X_p^{(k-1)}$
8. $\quad$ while($h > 0$)
9. $\quad\quad$ if(probe(message))
10. $\quad\quad\quad$ if(message = request$_{X_p}$)
11. $\quad\quad\quad\quad$ send(X_p, q)
12. $\quad\quad\quad$ else
13. $\quad\quad\quad\quad$ receive(X_q, q); $h \leftarrow h - 1$
14. $\quad\quad\quad\quad$ $Z \leftarrow Z - \check{A}_{pq} X_q^{(k-1)}$
15. $\quad$ serve(X_p, request$_{X_p}$)
16. $\quad$ for($0 \leq i < N_p$)
17. $\quad\quad$ $X_p^{(k)}{}_{[i]} \leftarrow D_{p[i]}{}^{-1} Z_{[i]}$
18. $\quad$ compute error collectively }

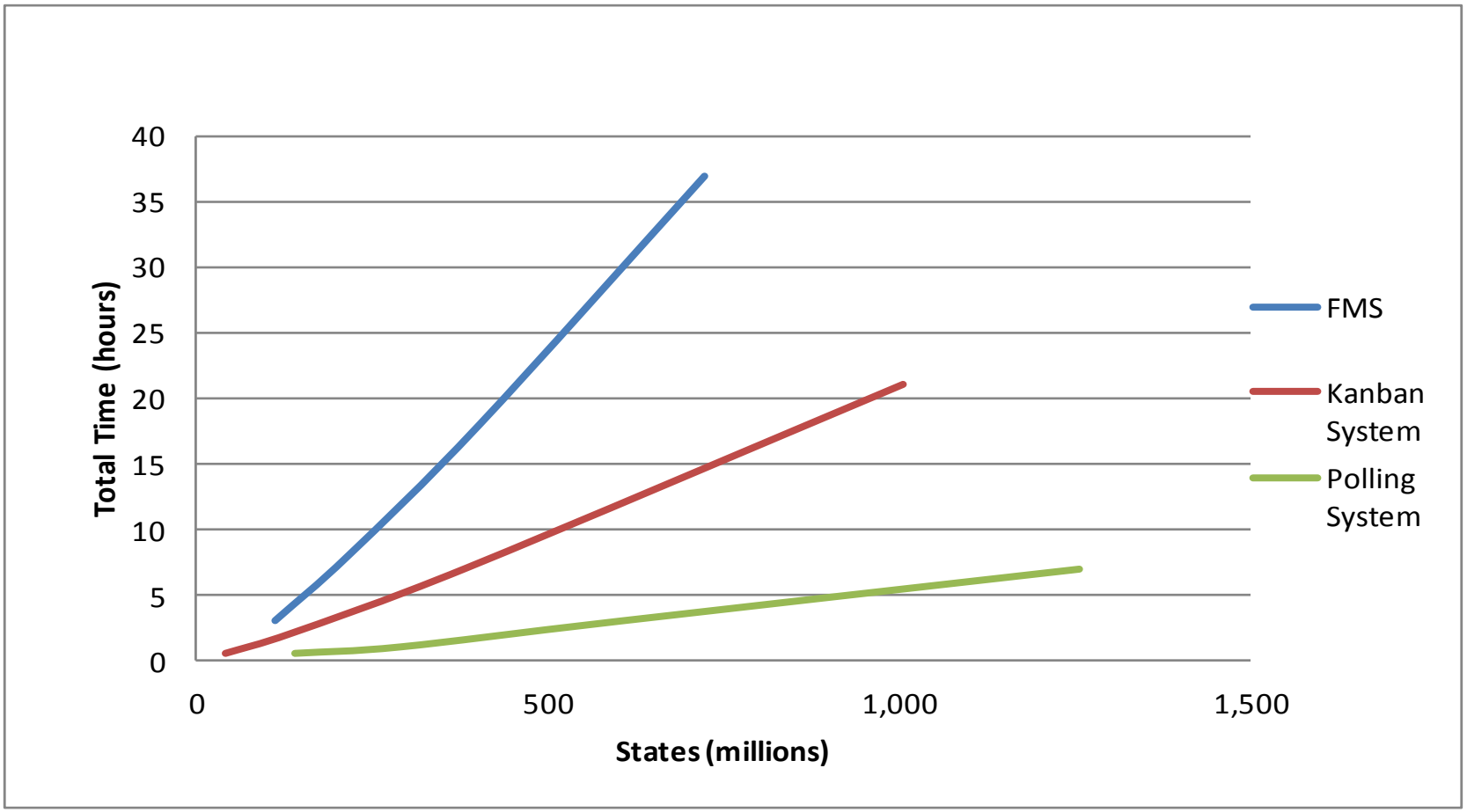
Parallel Execution on 48 Cores

k	States (*n*)	MB/Node	Times		Total Iterations
			Iteration (seconds)	Total (hr:min:sec)	
FMS Model					
12	111,414,940	170	6.07	3:40:57	2184
13	216,427,680	306	13.50	8:55:17	2379
14	403,259,040	538	25.20	18:02:45	2578
15	724,284,864	1137	48.47	37:26:35	2781
Kanban System					
7	41,644,800	53	1.73	33:07	1148
8	133,865,325	266	5.27	2:02:06	1430
9	384,392,800	564	14.67	7:03:29	1732
10	1,005,927,208	1067	37.00	21:04:10	2050
Polling System					
22	138,412,032	328	2.60	44:31	1027
23	289,406,976	667	5.33	1:36:02	1081
24	603,979,776	811	11.60	3:39:38	1136
25	1,258,291,200	1196	23.97	7:54:25	1190

Time Per Iteration



Total Time


40
35
30
25
20
15
10
5
0
Total Time (hours)
0
500
1,000
1,500
States (millions)
FMS
Kanban System
Polling System

Conclusions

- It is possible to solve large sparse systems in a *reasonable* time using out of core and parallel solutions
- Exploiting structure in matrices can be the key in some cases
- There are always possibilities to exploit knowledge of computing software/ hardware to bring innovation
- New and Emerging computing architectures are increasingly complex
- The complexity does offer many opportunities for new methods and innovation
- SSDs, RAIDs, manycore architectures, GPU, FPGAs offer great opportunities for out-of-core methods and parallelisation
- The real opportunity is to design intelligent software that is able to dynamically find the optimum schedule of parallelisation on a mix of architectures

Thank You

R.Mehmood@hud.ac.uk