# Solution Methods for the Analysis of Large Markov Chains (CTMCs)

*Rashid Mehmood*

Swansea University

MATHEMATICS OF NETWORKS

(Sixth Meeting)

29 June 2007

# Context

- Stochastic Modelling
  - unpredictability is inherent in most real-life systems
    - e.g. communication networks
- Tools for analyzing such systems:
  - e.g. discrete-state models
  - e.g. Markov Chains, CTMCs (continuous time)
- Behaviour of physical systems can be described by:
  - a set of states and
  - state to state transitions

# Context . . .

- Three steps in performance evaluation:
  - 1. specification
    - Model description:

      e.g. queueing networks, stochastic Petri nets, . . .
    - Properties specification:

      probabilistic logics (CSL, PCTL,. . . )
  - 2. state space generation
    - matrix generation from the formalism
  - 3. computing performance measures
    - e.g. solving $Ax = 0$ for steady state solution
- Problem: State-space explosion
  - we concentrate on phase three (*computationally expensive*):
    - numerical solution of $Ax = 0$

# State-Space Explosion

- The task is the iterative solution of $Ax = 0$
  - requires storage of the matrix $A$ and the vector(s) $x$
- Explicit techniques
  - disk-based storage of matrix (out-of-core)
  - distribution to parallel nodes
  - disk-based + parallel
- Implicit techniques (compact representation of matrix)
  - e.g. symbolic: BDD-based data structures
- The aim
  - extend the size of solvable models
  - develop new solution techniques
  - seek improvements in the existing
  - build realistic models of interesting applications

# Contents

- Numerical Computations and the available Methods
- Matrix Storage
  - Sparse and symbolic storage schemes
- The out-of-core solution Methods
- Parallel Symbolic implementation of Jacobi solution method
- Applications I have been looking at
- Summary and Future Work

# The Numerical Computations

- Steady state computations
  - $\pi Q = 0, \; \sum_{i=0}^{n-1} \pi_i = 1$
- Direct Methods
  - Modify the matrix
  - Problem: *fill-in*
- Iterative methods
  - Based on matrix vector products (MVP)

# Iterative Methods for $Ax = b$

- $Residual = b - Ax$
  - Make initial guess for the solution vector $x$
  - Generate successive approximations
  - Until residual falls below some prescribed value
  - Matrix is unchanged
- Basic Iterative methods
  - Jacobi, Gauss-Seidel, SOR
  - Iteratively, modify components of solution vector
  - Convergence tested for residual or error vector
- Projection methods
  - Extract an approximate solution $x$ from a subspace of $\mathbb{R}^n$
  - Orthogonalisation, nonstationary, parameter-free
  - Krylov subspace methods

# Transition Matrices

- Continuous Time Markov Chains
  - Transition rate matrix $Q$
    - $S \times S \rightarrow \mathbb{R}$
    - Diagonal entries: $q_{ii} = -\sum_{i \neq j} q_{ij}$
- Properties
  - Very large, sparse
  - Number of distinct values depends on the model

# Transition Matrices (Polling System)

- Case Study: Cyclic Sever Polling System [Ibe and Trivedi, 1990]
  - $k$ stations or queues and a server
  - server polls the stations in a cycle looking for jobs

| $k$ | *states* ($n$) | *off-diagonal nonzero* ($a$) | $a/n$ | MB per $\pi$ |
|---|---|---|---|---|
| 17 | 3,342,336 | 31,195,136 | 9.3 | 26 |
| 18 | 7,077,888 | 69,599,232 | 9.8 | 54 |
| 20 | 31,457,280 | 340,787,200 | 10.8 | 240 |
| 21 | 66,060,288 | 748,683,264 | 11.3 | 513 |
| 22 | 138,412,032 | 1,637,875,712 | 11.8 | 1,056 |
| 23 | 289,406,976 | 3,569,352,704 | 12.3 | 2,204 |
| 24 | 603,979,776 | 7,751,073,792 | 12.8 | 4,608 |
| 25 | 1,258,291,200 | 16,777,216,000 | 13.3 | 9,598 |

# Case Studies (Kanban and FMS Systems)

- Case Study: FMS [Ciardo and Tilgner , 1993]
  - Flexible Manufacturing System
  - models a system with three machines
  - machines process different types of parts
  - model parameter $k$: max. no. of parts machine can handle

- Case Study: Kanban [Ciardo and Tilgner , 1996]
  - Kanban Manufacturing System
  - models a system with four machines
  - parameter $k$ is the max. no. of machine jobs at a time

# Sparse Matrix Storage Schemes

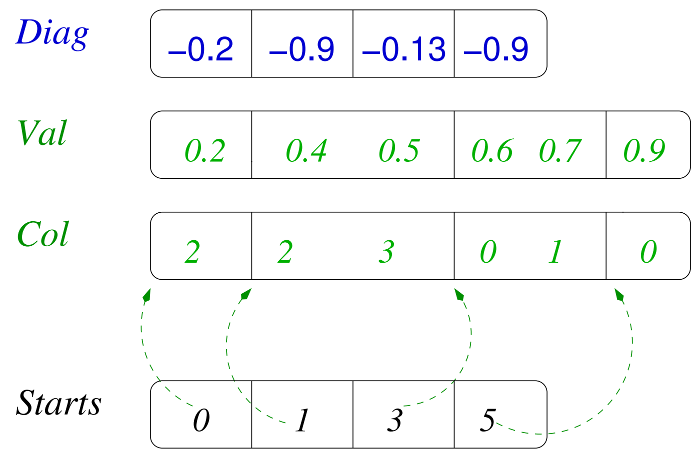
- Coordinate format, Compressed Sparse Row (CSR) schemes
  - Store whole matrix with no distinction between diagonal and off-diagonal entries

- Modified (compressed) Sparse Row (MSR)
  - Four arrays:
    - `double Diag`$[n]$
    - `double Val`$[a]$
    - `int Col`$[a]$
    - `int Starts`$[n]$
  - Bytes required: $12(a+n)$

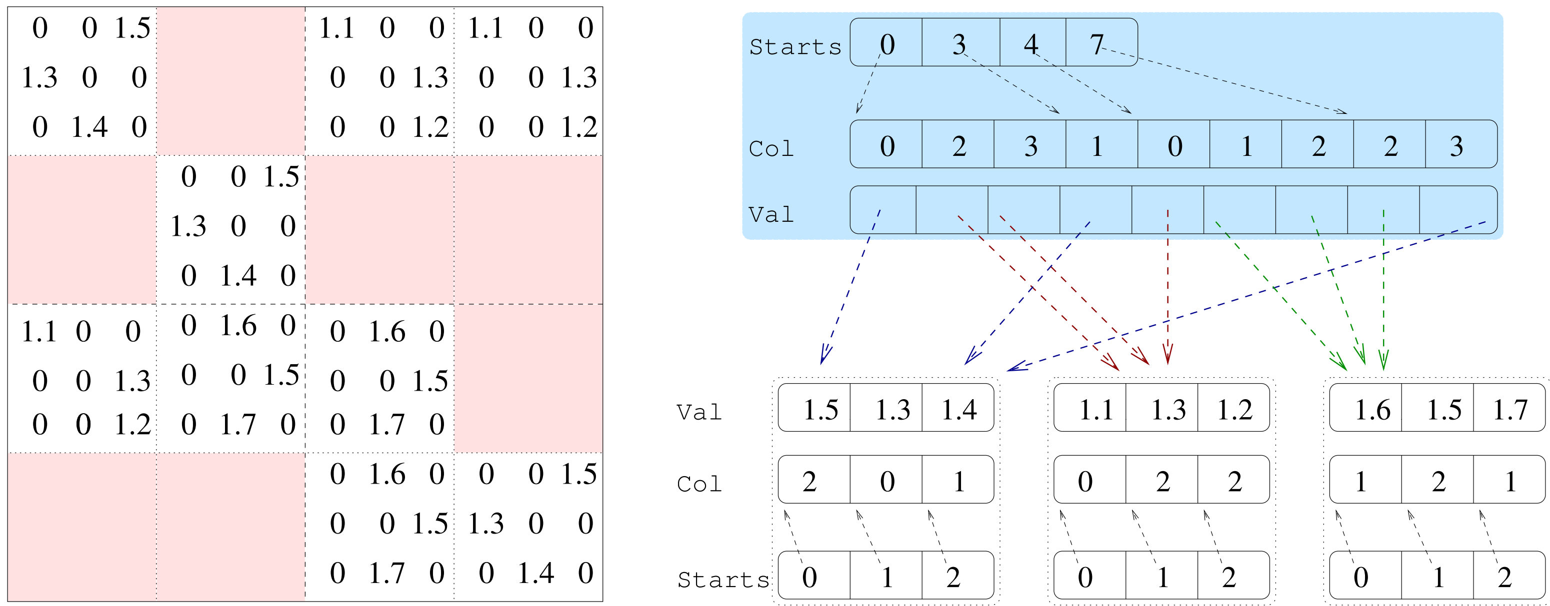
# Sparse Matrix Storage Schemes...

$$\begin{pmatrix} -0.2 & 0 & 0.2 & 0 \\ 0 & -0.9 & 0.4 & 0.5 \\ 0.6 & 0.7 & -0.13 & 0 \\ 0.9 & 0 & 0 & -0.9 \end{pmatrix}$$

$$n = 4, \;\; a = 6$$



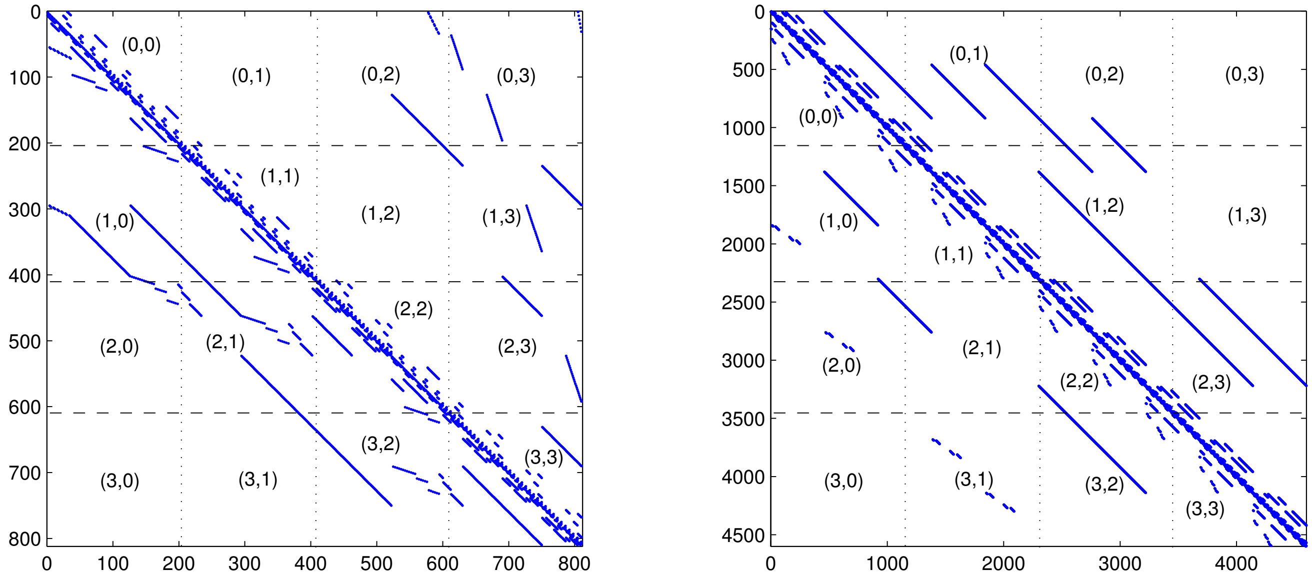
# The Symbolic Representation of a CTMC



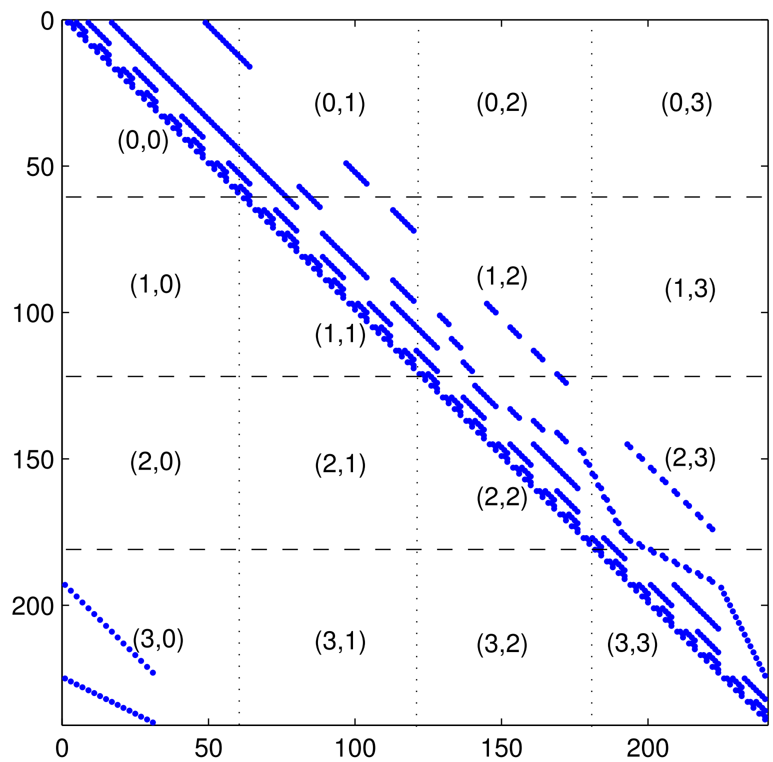
# Comparison of Storage Methods

| $k$ | States ($n$) | Off-diagonal nonzeros ($a$) | $a/n$ | Memory for Matrix (MB) *MSR* | *Ind. MSR* | *Comp. MSR* | *MTBDDs* | Vector (MB) |
|---|---|---|---|---|---|---|---|---|
| | | | | FMS models | | | | |
| 6 | 537,768 | 4,205,670 | 7.82 | 50 | 24 | 17 | 4 | 4 |
| 10 | 25,397,658 | 234,523,289 | 9.23 | 2,780 | 1,366 | 918 | 137 | 194 |
| 13 | 216,427,680 | 2,136,215,172 | 9.87 | 25,272 | 12,429 | 8,354 | 921 | 1,651 |
| 14 | 403,259,040 | 4,980,958,020 | 12.35 | 58,540 | 28,882 | 19,382 | 1,579 | 3,077 |
| 15 | 724,284,864 | 9,134,355,680 | 12.61 | 107,297 | 52,952 | 35,531 | 2,676 | 5,526 |
| | | | | Kanban models | | | | |
| 4 | 454,475 | 3,979,850 | 8.76 | 47 | 23 | 16 | 1 | 3.5 |
| 6 | 11,261,376 | 115,708,992 | 10.27 | 1,367 | 674 | 452 | 6 | 86 |
| 9 | 384,392,800 | 4,474,555,800 | 11.64 | 52,673 | 25,881 | 17,435 | 99 | 2,933 |
| 10 | 1,005,927,208 | 12,032,229,352 | 11.96 | 141,535 | 69,858 | 46,854 | 199 | 7,675 |
| | | | | Polling System | | | | |
| 15 | 737,280 | 6,144,000 | 8.3 | 73 | 35 | 24 | 1 | 6 |
| 21 | 66,060,288 | 748,683,264 | 11.3 | 8,820 | 4,334 | 2,910 | 66 | 504 |
| 24 | 603,979,776 | 7,751,073,792 | 12.8 | 91,008 | 44,813 | 30,067 | 144 | 1,136 |
| 25 | 1,258,291,200 | 16,777,216,000 | 13.3 | 196,800 | 96,960 | 65,040 | 317 | 1,190 |

# Sparsity Pattern (Kanban and FMS)



# Sparsity Pattern (Polling)



# The Computations

- A revisit: MVP operation is central to
  - Krylov subspace and basic iterative methods for steady state solution
  - uniformisation method for transient solution

- An in-core MVP-based Jacobi iteration

$r \leftarrow -Q^T \pi$

`for` $i = 0$ to $n-1$

$\quad \tilde{\pi}_i \leftarrow \pi_i + r_i / q_{ii}$

$r \leftarrow -Q^T \tilde{\pi}$

Test for convergence

$\pi \leftarrow \tilde{\pi}$

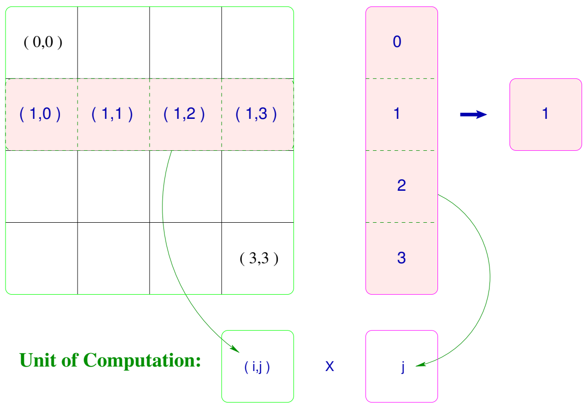
Repeat iteration if required

# The Computations...

- Storage requirement
  - Matrix storage in some sparse format
  - Two iteration vectors: $2 \times 8n$ Bytes
- Gauss-Seidel
  - Typically, converges faster than Jacobi
  - Requires one iteration vector
  - Standard MVP not possible!
  - Partitioning the matrix and vector into blocks can help

# 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 ($A_{ij} \times X_j$)



# A (Serial) Block Jacobi Algorithm

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

$\underline{\texttt{var}}\ \tilde{x},\ Y,\ k \leftarrow 0,\ \texttt{error} \leftarrow 1.0,\ i,\ j,\ p$
$\underline{\texttt{while}}(\ \texttt{error} > \varepsilon)$
$\quad k \leftarrow k + 1$
$\quad \underline{\texttt{for}}(\ 0 \leq i < P)$
$\qquad Y \leftarrow B_i$
$\qquad \underline{\texttt{for}}(\ 0 \leq j < P)$
$\qquad\quad Y \leftarrow Y - \check{A}_{ij} X_j^{(k-1)}$
$\qquad \underline{\texttt{for}}(\ 0 \leq p < n[i])$
$\qquad\quad X_i^{(k)}[p] \leftarrow D_i[p]^{-1} Y[p]$
$\quad$ compute $\texttt{error}$
$\quad X_i^{(k-1)} \leftarrow X_i^{(k)}$ }

# An 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*

```
Local variables: i, j, t = 0
while not converged
   for i ← 0 to B − 1
      for j ← 0 to B − 1
         if not an empty block
            disk_read (A_ij, R_t)
         Signal(S_1)
         Wait(S_2)
         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 on a 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 |

# A Parallel Jacobi Algorithm for Process $p$

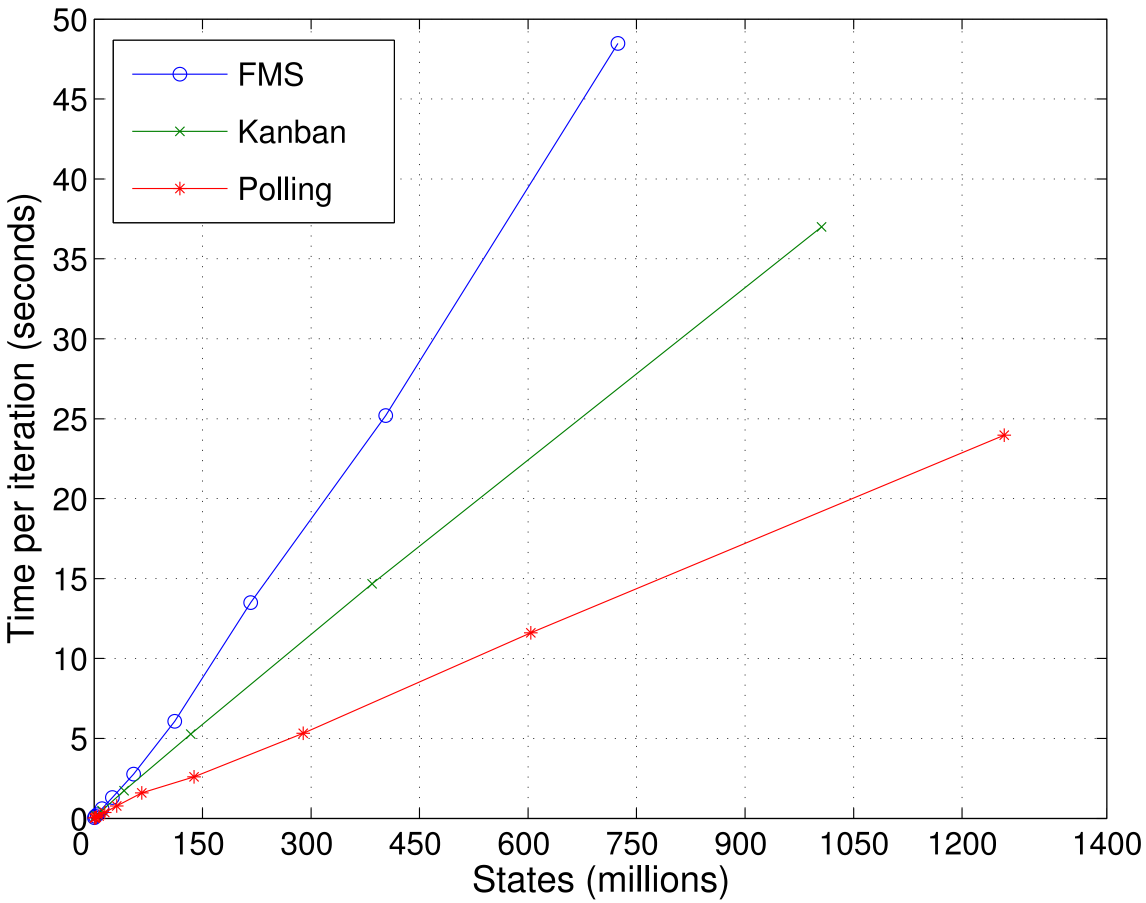
par_block_Jac( $\check{A}_p,\ D_p,\ B_p,\ X_p,\ T, N_p,\ \varepsilon$ ) {

$\underline{\texttt{var}}\ \tilde{X}_p,\ Z,\ k \leftarrow 0,\ \texttt{error} \leftarrow 1.0,\ q,\ h,\ i$
$\underline{\texttt{while}}(\ \texttt{error} > \varepsilon)$
$\quad k \leftarrow k+1;\ \ h \leftarrow 0$
$\quad \underline{\texttt{for}}(\ 0 \le q < T; q \ne p)$
$\quad\quad \underline{\texttt{if}}(\ \check{A}_{pq} \ne \mathbf{0})$
$\quad\quad\quad \underline{\texttt{send}}(\ \texttt{request}_{X_q}, q);\ \ h \leftarrow h+1$
$\quad Z \leftarrow B_p - \check{A}_{pp} X_p^{(k-1)}$
$\quad \underline{\texttt{while}}(\ h > 0)$
$\quad\quad \underline{\texttt{if}}(\ \underline{\texttt{probe}}(\ \texttt{message})\ )$
$\quad\quad\quad \underline{\texttt{if}}(\ \texttt{message} = \texttt{request}_{X_p})$
$\quad\quad\quad\quad \underline{\texttt{send}}(\ X_p, q)$
$\quad\quad\quad \underline{\texttt{else}}$
$\quad\quad\quad\quad \underline{\texttt{receive}}(\ X_q, q);\ \ h \leftarrow h-1$
$\quad\quad\quad\quad Z \leftarrow Z - \check{A}_{pq} X_q^{(k-1)}$
$\quad \underline{\texttt{serve}}(\ X_p, \texttt{request}_{X_p})$
$\quad \underline{\texttt{for}}(\ 0 \le i < N_p)$
$\quad\quad X_p^{(k)}[i] \leftarrow D_p[i]^{-1} Z[i]$
$\quad$ compute $\texttt{error}$ collectively }

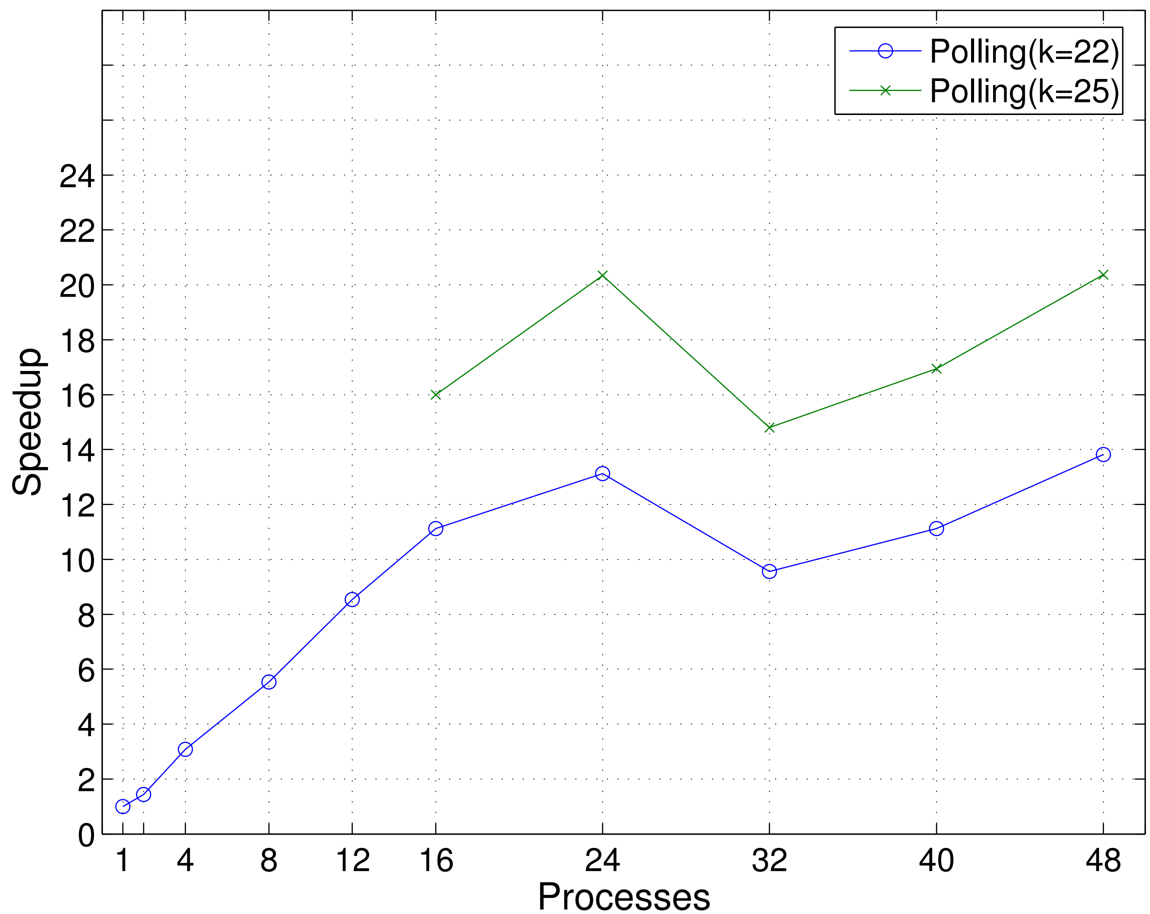
# Parallel Execution Results on 24 Nodes

| $k$ | States ($n$) | MB/Node | Time: Iteration (seconds) | Time: Total (hr:min:sec) | Total Iterations |
|---|---|---|---|---|---|
| | | | 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 |

# Comparison of Solution Times (Iteration)



# Speedups for the Polling System



# Applications I am looking at

- Debugging and managing distributed systems
    - event-based middleware
    - modelling program behaviour
    - integration of traditional debugging and model checking
    - scheduling and optimisation
    - systems security
        - intrusion detection through analysis
- Road traffic and networks
    - real data from M4 motorway
    - city and intercity traffic

# Applications I am looking at. . .

- Optical and storage area networks
  - WDM metro area ring networks
  - SAN extensions and mirroring strategies
- Mobile ad hoc and sensor networks
  - mobility, applications, network layer. . .
- Communications traffic modelling
  - multimedia and services
  - transport layer protocols

# Summary and Future Work

- State Space Explosion
  - discrete state models: CTMCs, DTMCs, MDPs
  - symbolic storage does help
  - out-of-core and parallel solutions
  - the largest models solved, over a billion states
- Apply the developed solution techniques to
  - more interesting real-life case studies
  - build realistic models
  - gain novel insight
- Parallel Algorithms
  - make it more adaptive
  - extend to Grids and ad hoc environments