ICASE Report #94-44 NASA Contractor Report #194929

On the Utility of the Multi-Level Algorithm for the Solution of Nearly Completely Decomposable Markov Chains

Scott T. Leutenegger [*]
Institute for Computer Applications in Science and Engineering
Mail Stop 132c, NASA Langley Research Center, Hampton, VA 23681-0001
leut@icase.edu

Graham Horton [†]
Lehrstuhl für Rechnerstrukturen, Universität Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Federal Republic of Germany
graham@immd3.informatik.uni-erlangen.de

Abstract

Recently the Multi-Level algorithm was introduced as a general purpose solver for the solution of steady state Markov chains. In this paper we consider the performance of the Multi-Level algorithm for solving Nearly Completely Decomposable (NCD) Markov chains, for which special-purpose iterative aggregation/disaggregation algorithms such as the Koury-McAllister-Stewart (KMS) method have been developed that can exploit the decomposability of the the Markov chain. We present experimental results indicating that the general-purpose Multi-Level algorithm is competitive, and can be significantly faster than the special-purpose KMS algorithm when Gauss-Seidel and Gaussian Elimination are used for solving the individual blocks.

[*]This research was supported by the National Aeronautics and Space Administration under NASA contract NAS1-19480 while the second author was in residence at the Institute for Computer Applications in Science and Engineering, NASA Langley Research Center.

[†]This work was carried out in part while the second author was a guest at ICASE, NASA Langley Research Center.

1 Introduction

Recently the Multi-Level (ML) algorithm was introduced as a fast solver for general steady-state Markov chains [2]. The algorithm has been shown to be faster, often one or two orders of magnitude faster relative to the Gauss-Seidel and optimized SOR algorithms for a number of different Markov chains. The study showed the potential of the algorithm, but in the light of the small number of test problems represented only a first step in determining whether the excellent performance can be realized for all Markov chains. In this work we investigate the utility of the ML algorithm for solving Nearly Completely Decomposable (NCD) Markov chains. For this class of Markov chains special-purpose iterative aggregation/disaggregation (IAD) algorithms [3, 11] have been shown to perform well. Thus, demonstration of the utility of an algorithm for solving NCD chains necessitates comparison with one of these IAD schemes. We include the Koury-McAllister-Stewart (KMS) [3] algorithm as a representative example. We also include the Gauss-Seidel (GS) method for purposes of comparison.

In our experiments we structure the Markov chains in NCD "normal", i.e. almost block-diagonal form. Hence we give an advantage to KMS, since in practice the algorithm would require computation time for the restructuring of the Markov chain. However, Stewart and Wu have shown that this restructuring time often is not significant [9]. Like Stewart and Wu, we use Gaussian elimination to solve blocks of size less than 100 states.

One difference between our experiments and those of Stewart and Wu is that they use SOR as well as GS to solve blocks. Note that we have found using SOR with a well chosen relaxation parameter for solving the blocks can decrease the computation time of the KMS algorithm by a factor of 3 or more for many of the chains we consider, but determining the correct parameter is not a trivial task. Another difference is that when Stewart and Wu use SOR or GS to solve a block, they perform a fixed number of iterations on the block whereas we perform iterations until the solution does not improve by more than ϵ in any state, for some ϵ.

The rest of the paper is organized as follows. In section 2 we provide some background material on NCD chains. In sections 3 and 4 we briefly describe the ML and KMS algorithms. Section 5 contains our experimental results. The final section contains our conclusions.

2 Nearly Completely Decomposable Markov Chains

In this section we give a brief informal description of NCD chains and give two examples that we use in our experimental studies. The seminal work on NCD systems was conducted by Simon and Ando [6], and applied to Markov chains by Courtois [1]. For a more detailed treatment we refer the reader to Stewart [9, 10]. An NCD Markov chain with B blocks can be ordered so that the generator matrix can be written in block form as:

$$Q = \begin{pmatrix} Q_{11} & Q_{12} & \dots & Q_{1B} \\ Q_{21} & Q_{22} & \dots & Q_{2B} \\ . & . & . & . \\ . & . & . & . \\ Q_{B1} & Q_{B2} & \dots & Q_{BB} \end{pmatrix} , \tag{1}$$

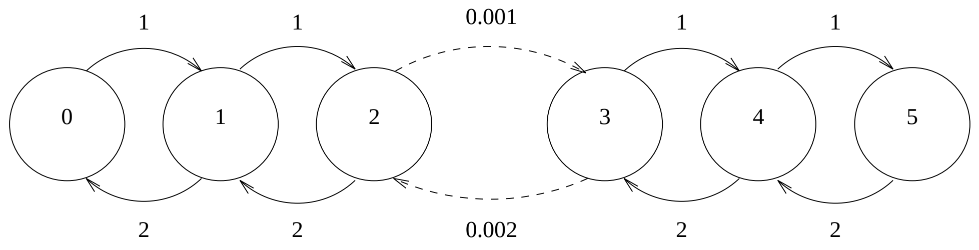

Figure 1: Sample Nearly Completely Decomposable Birth-Death Markov Chain

where the elements of the off-diagonal blocks $Q_{ij},\ i \neq j$ are small compared to the elements of the diagonal blocks Q_{ii}. The larger the elements of the off-diagonal blocks, the less NCD the chain becomes. Note that for a completely decomposable chain the off-diagonal blocks would contain only zeros and the Markov chain degenerates into a set of independent problems. Any solution algorithm that makes use of the block structure will have to invest some pre-processing effort into permuting the matrix into the above almost block-diagonal form.

In this paper we consider two different example NCD Markov chain structures.

2.1 Example 1

The first example is a modified birth-death chain that is subdivided into B blocks of size N. The birth rate between blocks is $\delta\times$ the birth rate within each block, and the death rate between blocks is $\delta\times$ the internal death rate. The parameter δ thus controls the degree of decomposability of the chain. In figure 1 we provide an example with 2 blocks, three states per block, a birth (death) rate of 1 (2), and $\delta = 0.001$. In our experiments we will vary both the number of blocks for a given chain size and the inter-block coupling strength δ.

2.2 Example 2

The second example NCD Markov chain structure, shown in figure 2, is the interactive computer system model used by Stewart [4, 8, 9]. In this model a CPU processes jobs arriving from a set of terminals, which may subsequently require service from a filing device (FD) or secondary memory (SM). Let L_x equal the length at queue "x". Let η equal the number of jobs in the system, giving $\eta = L_{CPU} + L_{FD} + L_{SM}$. We assume there is one job per terminal, and that the terminals are modeled as an infinite server. The Markov chain is decomposed into $B + 1$ blocks, where B is the number of terminals and block i is the set of all states which have i jobs in the system. These blocks form an NCD chain when the terminal service rate and the probability of returning to the terminals after visiting the CPU are small.

We assume the same parameter values as are used in [9]: the rate leaving the terminals is $\frac{m}{10{,}000}$, the rate leaving the SM device is 0.2, the rate leaving the FD is $\frac{1}{30}$, the rate from the CPU to the FD is 0.05, the rate from the CPU to the terminals is 0.002, and the rate from the CPU to the SM device is $100(\frac{\eta}{128})^{1.5}$. This last rate is meant to model the rate at which page faults occurs and increases with the number of currently executing jobs. For a more detailed description of the model we refer the reader to

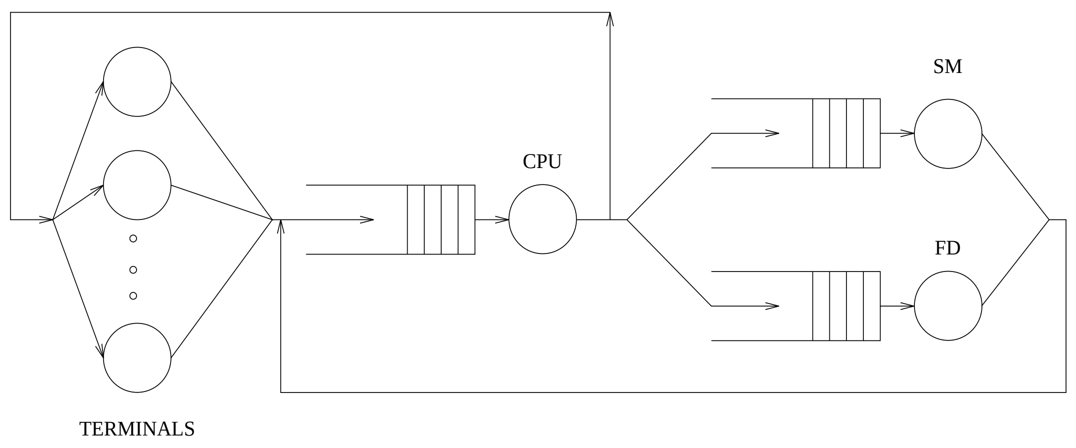

Figure 2: Multiprogrammed Queueing Network Model

[9, 4].

3 Multi-Level Algorithm

In this section we briefly review the recently introduced multi-level algorithm, details of which can be found in [2]. Consider a Markov chain consisting of n states $s_0 \ldots s_{n-1}$. Denote the unknown vector by p, where p_i is the probability of finding the Markov chain in state s_i.

We then have to solve the system of equations

$$Pp = 0 \tag{2}$$

with the additional condition

$$\sum_{i=0}^{i=n-1} p_i = 1 \tag{3}$$

This equation is usually written as $\pi Q = 0$ for $\pi = p^T$ and $Q = P^T$, the infinitesimal generator matrix.

A coarser representation of the Markov chain described by matrix P may be obtained by *aggregation*. This means creating a new Markov chain described by a matrix Q with the vector of state probabilities q, each of whose N states $S_0 \ldots S_{N-1}$ is derived from a number of states of the original system.

In the following we will use the terms *fine level* and *coarse level* to refer to Markov chains where the latter is obtained by aggregation from the former. The relation $s_k \in S_i$ signifies that the fine level state s_k is mapped by the aggregation operation to the coarse level state S_i.

The matrix Q of the aggregated system is chosen as follows :

$$Q_{ji} = \frac{\sum\limits_{s_k \in S_i} p_k \sum\limits_{s_l \in S_j} P_{lk}}{\sum\limits_{s_k \in S_i} p_k} \tag{4}$$

```
procedure ml(l)
    if ($l = 0$)
        solve $P_l \bar{p}_l = 0$
    else
        $\tilde{p}_l$ = GS$^\nu$($\bar{p}_l$)
        $\tilde{p}_{l-1}$ = $R_{l-1,l}(\tilde{p}_l)$
        ml($l-1$)
        $p^*_{l-1}$ = $\bar{p}_{l-1}/\tilde{p}_{l-1}$
        $p^*_l$ = $I_{l-1,l}(p^*_{l-1})$
        $\bar{p}_l$ = $C(\tilde{p}_l, p^*_l)$
    return
```

Figure 3: Multi-Level Algorithm

This is the well-known *aggregation matrix.* Note that the matrix Q is a function not only of the fine level matrix P, but also of the fine level solution vector p.

This yields the aggregated equations in the unknown q:

$$\begin{aligned} Qq &= 0 \\ \sum_{i=0}^{N-1} q_i &= 1 \quad . \end{aligned}$$

It can then be shown that

$$q_i = \sum_{s_k \in S_i} p_k \quad , \tag{5}$$

i.e. the solution q of the aggregated system truly represents a coarser version of the solution p of the original problem. The probability of being in state q_i is the sum of the probabilities of being in any of its constituent fine-level states. We use the aggregation equation as a basis for the multi-level method, whereby we approximate the exact solution values p_k in (4) above by values from the current iterate.

The ML algorithm can be viewed as a recursive aggregation/disaggregation algorithm. We adopt the following abbreviations for vectors $a,\ b,\ c \in I\!R^m$:

$$\begin{aligned} a = b * c &\equiv a_i = b_i * c_i, \quad 1 \le i \le m \\ a = b/c &\equiv a_i = b_i / c_i, \quad 1 \le i \le m \end{aligned}$$

The *two*-level version of the ML iteration is given by the following sequence of steps.

- Perform Gauss-Seidel relaxation on current iterate $p^{(i)}$, yielding $\tilde{p}$

$$\tilde{p} = GS(p^{(i)}) \tag{6}$$

- Restrict current solution $\tilde{p}$ to coarse-level vector $\tilde{q}$

$$\tilde{q} = R(\tilde{p}) \quad \equiv \quad \tilde{q}_i = \sum_{s_k \in S_i} \tilde{p}_k \tag{7}$$

- Compute coarse matrix $\tilde{Q}$

$$\tilde{Q}_{ji} = \frac{\sum\limits_{s_k \in S_i} \tilde{p}_k \sum\limits_{s_l \in S_j} P_{lk}}{\tilde{q}_i} \tag{8}$$

- Solve coarse equation for $\bar{q}$

$$\tilde{Q}\bar{q} = 0 \ , \quad \sum_{i=0}^{N-1} \bar{q}_i = 1 \tag{9}$$

- Compute coarse level correction q^*

$$q^* = \bar{q}/\tilde{q} \tag{10}$$

- Compute fine level correction p^*

$$p^* = I(q^*) \quad \equiv \quad p_k^* = q_i^* \tag{11}$$

- Apply fine level correction to obtain new iterate $p^{(i+1)}$

$$p^{(i+1)} = \bar{p} = C(\tilde{p}, p^*) \quad \equiv \quad \tilde{p} * p^* \tag{12}$$

In this two-level form the method is similar to well-known iterative aggregation/disaggregation (IAD) methods such as those of Koury, McAllister and Stewart [3] and of Takahashi [11]. The *multi*-level algorithm is obtained by recursive application of the two-level algorithm to obtain a solution to the aggregated equation (9) and is described in algorithmic form in figure 3. We use the subscript l to denote level of representation ($l = lmax$ finest level, i.e. the original Markov chain, $l = 0$ coarsest level). The coarse level $l-1$ and fine level l between which the operators I and R map are identified by appropriate indices. Note that, because of the recursive nature of the algorithm, the unknowns q^*, $\bar{q}$ and $\tilde{q}$ are represented by the variables p^*_{l-1}, $\bar{p}_{l-1}$ and $\tilde{p}_{l-1}$, respectively. We allow in general the possibility of applying `GS` ν times at each level with $\nu \geq 1$, denoted by $\texttt{GS}^{\nu}$.

The performance of the policy can be greatly influenced by the aggregation strategy used. The aggregation strategy used for all experiments in this paper attempts to aggregate pairs (or triples) of fine level states that are strongly coupled. Let Ψ be the maximum number of fine level states allowed to be aggregated into a single coarse level state. For all experiments in this paper Ψ is set to three. Let α be the aggregation rate differential, used for determining whether two states are coupled strongly enough to aggregate them together. For all experiments in this paper we set $\alpha = 0.5$. We loop though all states of a given level and for each state s_i that has not yet been assigned to an aggregated state:

1. Let $absolute_{max}$ = the maximum of $\{P_{ij}, P_{ji}\}$ $\forall j$.
2. Let $available_{max}$ = the maximum of $\{P_{ij}, P_{ji}\}$ $\forall j$ such that s_j has not yet been assigned to an aggregated state whose number of constituent fine level states is equal to Ψ.
3. If $available_{max} \geq (\alpha * absolute_{max})$, then aggregate s_i with s_j. Note, it is possible that s_j is already in an aggregate of size $\Psi - 1$ in which case s_i is added to the aggregate to make it of size Ψ.

4. If $available_{max} < (\alpha * absolute_{max})$, then map s_i to a new coarse level state by itself. Another state visited latter in the aggregation step may be aggregated with this state, if not the coarse state remains composed of only this one fine level state.

The goal of this aggregation method is to avoid coupling weakly connected states when in the presence of a strong connection to some other state, and also to avoid the creation of large aggregates. The parameter α ensures that states are strongly connected enough before aggregating them together, and the use of singletons avoids creation of large aggregates. This aggregation strategy takes strongly differing rates into account and therefore will aggregate within blocks when the chain is NCD. As a result, strongly coupled states are aggregated together and at some intermediate level the ML aggregation will likely correspond to the KMS coarse level.

4 KMS Algorithm

Iterative Aggregation/Disaggregation (IAD) methods are a well-known class of algorithms for the steady state solution of Markov chains and which bear a close relationship to the ML method presented here. IAD methods are reviewed in [7]. The generic IAD method is that of Koury, McAllister and Stewart (KMS) [3]. Using the notation of section 3, the KMS method is defined as follows:

1. Construct coarse level matrix $\tilde{Q}$ using (8)
2. Solve coarse system (9).
3. Perform the correction (11), (12).
4. For each coarse level state S_i solve the set of equations connecting all fine level states s_k aggregated to it ($s_k \in S_i$).
5. If not converged goto 1

Step (4) is a Block Gauss-Seidel step on the finer level, where the blocks are defined by the aggregated system corresponding to (1).

We now see that the KMS algorithm is a special case of ML, obtained by the following choices:

1. Use of only two levels of representation of the system, rather than recursively coarsened problems.
2. The number of fine unknowns aggregated into a single state is large, whereas for ML the number of unknowns is between one and three, usually two or three.
3. Use of Block Gauss-Seidel on the finer level, as opposed to a pointwise Gauss-Seidel scheme.

One open question regarding the KMS algorithm is how much work per global iteration should be done solving each block on the finer level. In Stewart [10, 9] SOR was applied for a fixed number of iterations. We chose instead to solve with GS until the solution does not improve by more than ϵ in any state, for ϵ values of 1e-06, 1e-09, and 1e-11.

Gaussian elimination for the solution of the blocks is generally faster than solution by Gauss-Seidel iteration when implemented in a way that preserves the sparsity of the matrices involved and these are of suitably small dimension. For larger blocks both the memory and cpu requirements force the switch to the iterative scheme. On the other hand, when the chain is NCD the KMS algorithm usually obtains a fairly accurate solution after the first iteration, meaning that Gauss-Seidel already has a good initial guess and requires only a few iterations to convergence. In this case iteration is to be preferred over elimination. As done in [9], we use Gaussian elimination for all blocks comprised of fewer than 100 states.

IAD methods such as this suffer the drawback that they are only applicable to NCD Markov chains. The aggregation is then defined by assigning one coarse unknown to each block. The number of fine unknowns that are aggregated to a single coarse state may be quite large.

5 Experimental Results

In this section we describe the experiments and the results obtained. In all figures, the lines are plotted in the same order they appear in the legends. We first present results from the generalized birth death structure described in section 2.1. This trivial example allows us to investigate how changing the number of states per block and decreasing the decomposability affect the relative performance of the policies. Note, in practice this chain would be solved analytically, we only solve it numerically to explore the relative solution speed of the algorithms.

For both experiments we fix the number of states in the chain at 1000, and set the birth (death) rate to be 50000 (40000). We choose large rates to prevent numerical underflow. In all figures we include data points for five algorithms: GS, ML, KMS-06, KMS-09, and KMS-11, where KMS-xx denotes the KMS algorithm assuming that block GS is applied on the blocks until the solution does not improve by more than 1e-xx in any state. We also ran the experiments using KMS-04, not shown, and the performance is very close to that of KMS-06.

In the first experiment we determine what effect the number of states per block has on the performance of the policies while keeping the problem size fixed. This is done by setting $\delta = 0.01$ and varying the number of blocks since a small number of blocks results in a large number of states per block. For this experiment we do not use Gaussian elimination for solving any of the blocks in KMS since this would prohibit us from attributing results to a single factor. If we used Gaussian elimination for small blocks the algorithm would change as the number of blocks changes, thus making the results difficult to interpret.

In figure 4 we plot the results. We include plots for the number of iterations, the overall floating point operation count in millions of floating point operations (MFLOPS), the time in seconds, and the ratio of flops relative to the ML algorithm. The time metric was obtained using the Unix *times* system call. The time metric results in similar comparisons, but is dependent on the efficiency of the implementation of the algorithms. In addition, we have found the timings to fluctuate do to interference form other jobs on the workstation, hence we focus on flops and iterations in subsequent experiments.

First consider the number of iterations needed. The behavior of KMS-11 is exactly what we would expect. KMS-11 requires a very small number of global iterations since the chain is strongly NCD and the individual blocks are solved to a high level of accuracy. The number of global iterations increases as the required accuracy of solving the individual blocks decreases. Note that the ML algorithm requires more iterations than KMS-11 but less than the other algorithms, and that GS requires a large number of iterations.

Now consider the flops metric. The ML algorithm requires significantly less computation than any of the other algorithms and the ordering of the KMS policies is reversed when considering flops instead of iterations. The KMS-11 algorithm may require fewer global iterations than the other KMS algorithms, but each iteration requires enough additional work to outweigh the reduced number of iterations. Note that Stewart [9, 10] pointed out that increasing the number of iterations when solving each block may or may not decrease overall solution time. In our experiments we find as a general rule that decreasing the accuracy required of each individual block solve decreases the overall solution time.

The performance of the KMS algorithm decreases as the block size increases because both block solvers considered (GS and Gaussian elimination) have a superlinear increase in operation counts in the number of unknowns. The saving per block outweighs the additional number of blocks now needed to be solved. The blocks used in this experiment were only of very modest size, ≤ 500. As the blocks increase in size ML becomes faster with respect to GS or Gaussian elimination, thus the superiority of ML will increase with the system size.

We now explore the sensitivity of the algorithms to the degree of decomposability of the Markov chain by varying the parameter δ. Our implementation of the KMS algorithm always assumes the same block structure regardless of δ, hence by increasing parameter δ we can decrease the decomposability. We plot the results in figure 5. The performance of the KMS-11 and KMS-9 algorithms quickly degrades as δ increases. The is not unexpected since the policy is designed to work on an NCD chain. The fact that it converges for high values of δ is encouraging. Note that KMS-6 is less sensitive to δ. The ML algorithm is almost completely insensitive to δ. This too is not particularly surprising since we previously showed the excellent performance of the ML algorithm for non-NCD chains [2].

We now consider the less trivial Markov chain of the queueing network model of the multiprogrammed computer system described in section 2.2. For these experiments we assume all KMS algorithms use Gaussian elimination for solving blocks with fewer than 100 states. The number of states per block is presented in table 1. For a population of 20 there are 21 blocks, for a population of 50 jobs there are 51 blocks.

In figure 6 we plot the results of the algorithms solving this model for a population of 20 when the rate from the cpu to the terminal is varied from 0.001 to 0.02. Note that a value of 0.002 was used in the experiments found in [9]. Our original intention in varying the rate from the cpu to the terminals was to determine how a change in the decomposability of the Markov chain affects the relative performance of the algorithms. The general trend noted from the number of flops is that as the rate is increased, hence the decomposability decreased, the KMS algorithms requires significantly less computation. This is counter-intuitive. The reason can be found in the solution values of the Markov chains. In figure 7 we plot the probability mass functions for the solution on a block basis. These plots were obtained by summing the probability mass over all states within each of the 21 blocks. As the rate of going from the cpu to the terminals is increased, probability mass shifts from the blocks with a large number of

1	3	6	10	15	21	28	36	45	55	66
78	91	105	120	136	153	171	190	210	231	253
276	300	325	351	378	406	435	465	496	528	561
595	630	666	703	741	780	820	861	903	946	990
1035	1081	1128	1176	1225	1275	1326				

Table 1: Sizes of the 51 Blocks for Example 2

states (corresponding to a large number of active jobs) to the blocks with a small number of states (corresponding to a small number of active jobs). Hence, the problem becomes trivial for the KMS algorithm to solve since the blocks with a large number of states require only one GS iteration. Thus, we find the performance of the KMS algorithm is very dependent on the solution to the Markov chain if the solution results in blocks whose probability mass is significantly smaller than the tolerance used for determining the stopping point of solving each block.

In figure 8 we plot the results assuming a population of 50 jobs. At a cpu to terminal rate of 0.001 most of the probability mass in the solution vector is in the last few blocks, hence KMS performs significantly worse than ML. For higher rates the mass shifts to the blocks with a small number of states and the performance of the KMS policies improves as in figure 6.

6 Conclusions

Our experimental evidence indicates that the ML algorithm is an efficient solver for NCD Markov chains. The algorithm is also insensitive to the degree of decomposability and the size of the blocks in the chain for the examples considered. These results strengthen the case that the ML algorithm is an efficient general purpose solver.

Our results also indicate that for the problems we have considered, the ML algorithm is faster than the KMS algorithm in its proposed form, i.e. using GS (Gaussian elimination) for solving blocks when the number of states in the block is ≥ 100 (≤ 100). We attribute the superior performance of the ML algorithm to the fact that the ML algorithm is much faster than GS or Gaussian elimination for solving large blocks.

At first it may seem surprising that the ML algorithm, not explicitly designed for NCD chains, should work so well. They key to understanding the good performance of the ML algorithm is the aggregation strategy. Since the algorithm aggregates strongly connected states together it has a similar effect as the IAD algorithms. In fact, for some small examples of our generalized birth death chain we have verified that the aggregated chain at some coarse level is the same chain found at the coarse level of the KMS algorithm.

A secondary contribution of this work is the discovery that the solution speed of the KMS algorithm (and we assume all traditional iterative aggregation/disaggregation algorithms) can be strongly affected by the solution *values* of the Markov chain. Note that the ML algorithm is much less sensitive to the solution values of the Markov chain.

The KMS algorithm is an attractive algorithm since it easily parallelizes and allows the efficient solution of problems that are larger than memory when the chain is NCD. We conclude therefore that if KMS is to be used, the ML algorithm should be used to solve the individual blocks since ML is a better solver for large blocks than GS, Gaussian elimination, or SOR. We plan to investigate this merger of the two algorithms in the near future.

References

[1] P. J. Courtois: **Decomposability; Queueing and Computer System Applications,** Academic Press, Orlando, FL.

[2] G. Horton, S. Leutenegger: *A Multilevel Solution Algorithm for Steady-State Markov Chains*, Proceedings of the ACM SIGMETRICS 1994 Conference on Measurement and Modeling of Computer Systems, Nashville, TN , May 16-20, 1994.

[3] R. Koury, D. McAllister, W. Stewart: *Methods for Computing Stationary Distributions of Nearly Completely Decomposable Markov Chains..* SIAM J. Alg. Disc. Math. Vol 5, No 2, 1984, pages 164-186.

[4] B. Philippe, Y. Saad, W. Stewart: *Numerical Methods in Markov Chain Modeling,* Operations Research, Vol 40, No 6, 1992.

[5] J. Ruge, K. Stüben: *Algebraic Multigrid.* In S. McCormick (Ed.) **Multigrid Methods.** SIAM, 1987.

[6] H. A. Simon, A. Ando: *Aggregation of Variables in Dynamic Systems,* Econometrics, Vol 29, p. 111-138, 1961.

[7] P. Schweitzer: *A Survey of Aggregation-Disaggregation in Large Markov Chains.* In W. Stewart **Numerical Solution of Markov Chains**, Marcel Dekker, 1991.

[8] W. J. Stewart: *A Comparison of Numerical Techniques in Markov Modelling*, Communications of the ACM, Vol 21, No 2, p. 144-152, 1978.

[9] W. J. Stewart, W. Wu: *Numerical Experiments with Iteration and Aggregation for Markov Chains*, ORSA Journal on Computing, Vol 4, No 3, 1992.

[10] W. J. Stewart: **Introduction to the Numerical Solution of Markov Chains**, book in progress.

[11] Y. Takahashi: *A Lumping Method for Numerical Calculations of Stationary Distributions of Markov Chains,* Research Report No. B-18, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 1975.

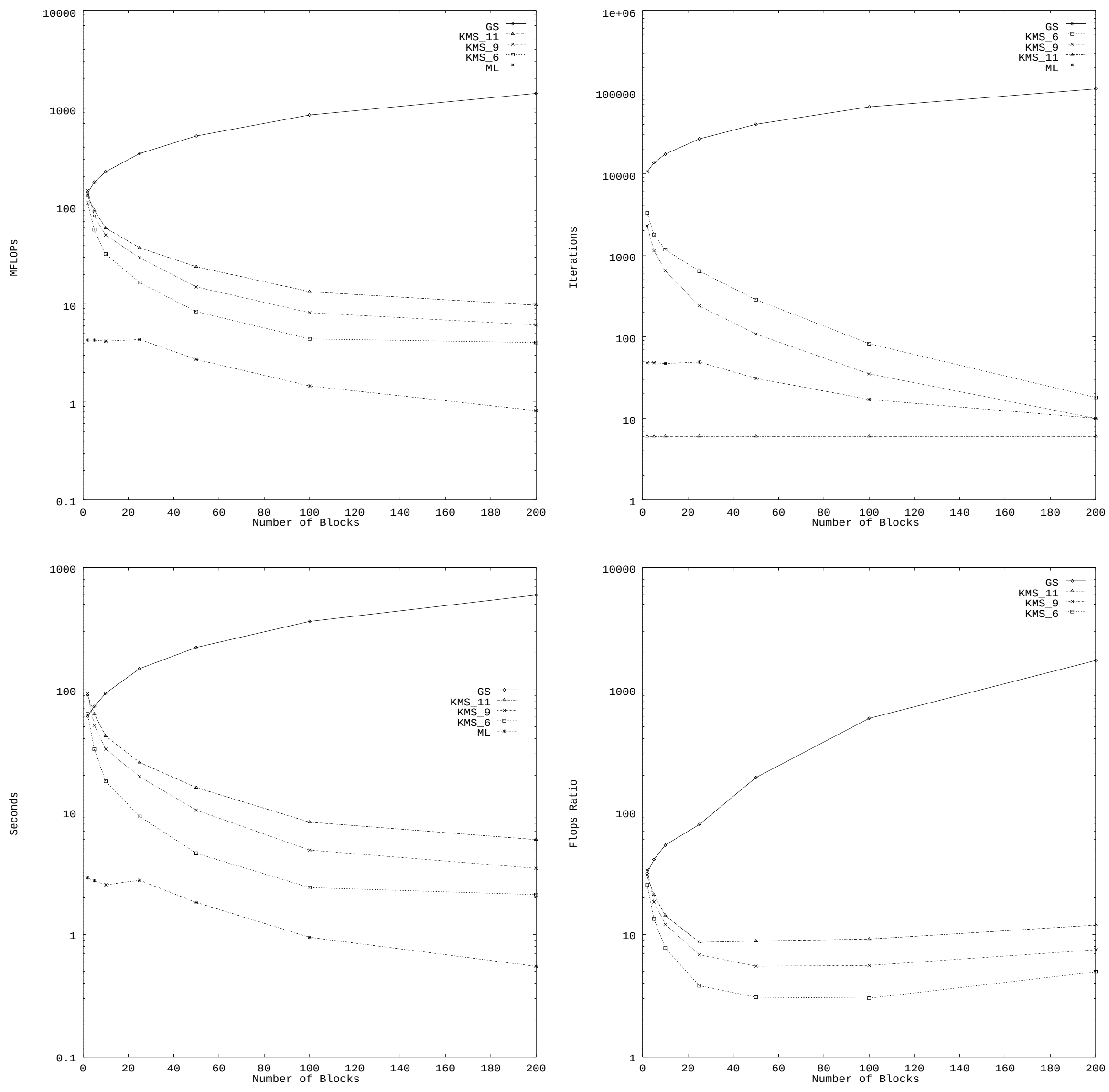

Figure 4: NCD Birth-Death chain. 1000 states, Number of blocks varied. Upper Left: MFLOPs; Upper Right: Number of Iterations; Lower Left: Computation time; Lower Right: ratio of FLOPs to ML.

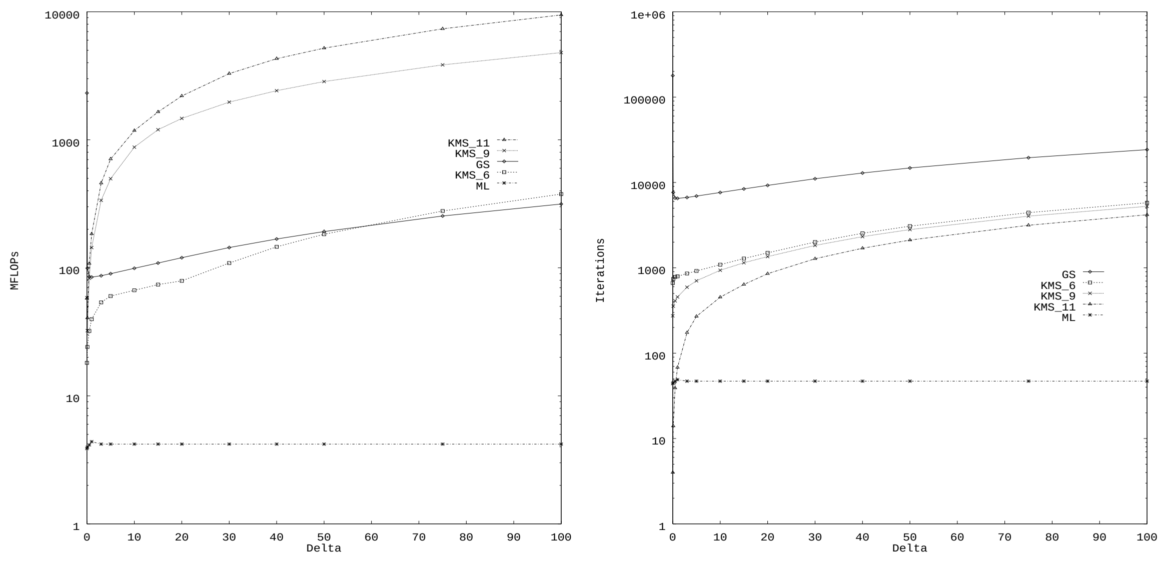

Figure 5: NCD Birth-Death chain. 1000 states, 20 blocks, δ varied. Left: MFLOPs; Right: Number of Iterations.

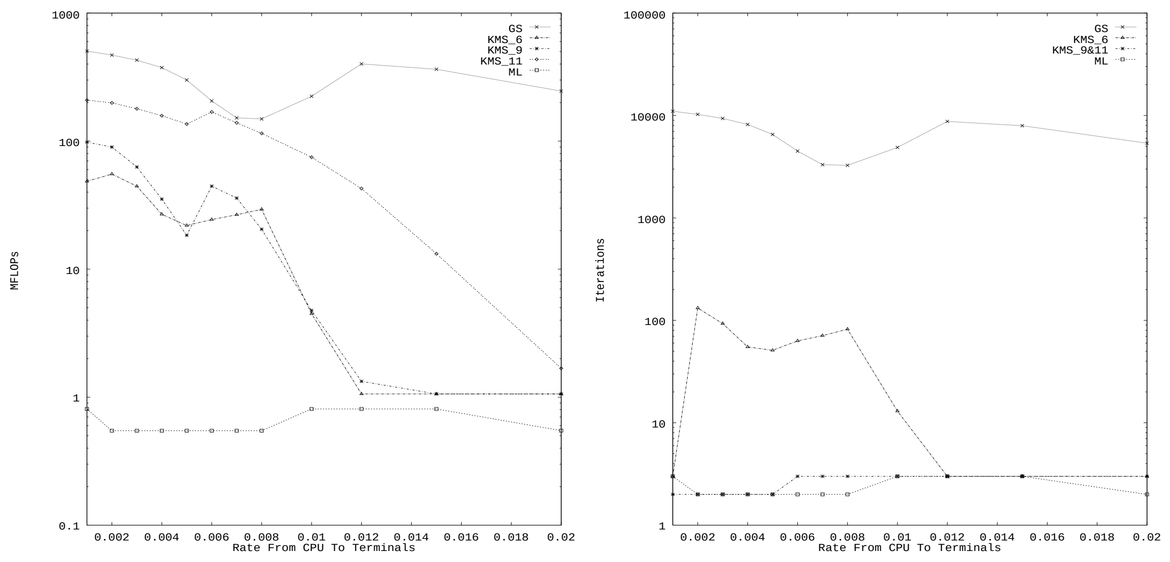

Figure 6: Interactive Computer System Model, 20 jobs, processing rate varied. Left: MFLOPs; Right: Number of Iterations.

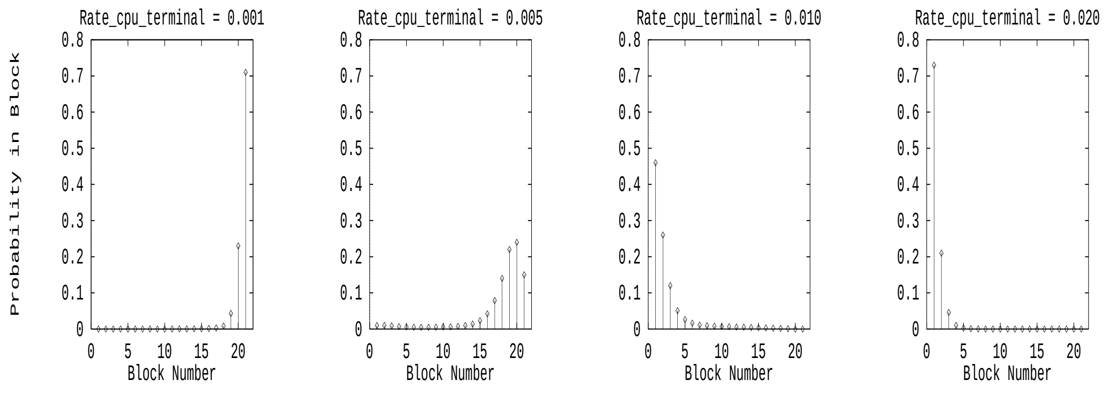

Figure 7: Interactive Computer System Model 20 Jobs, selected block probabilities

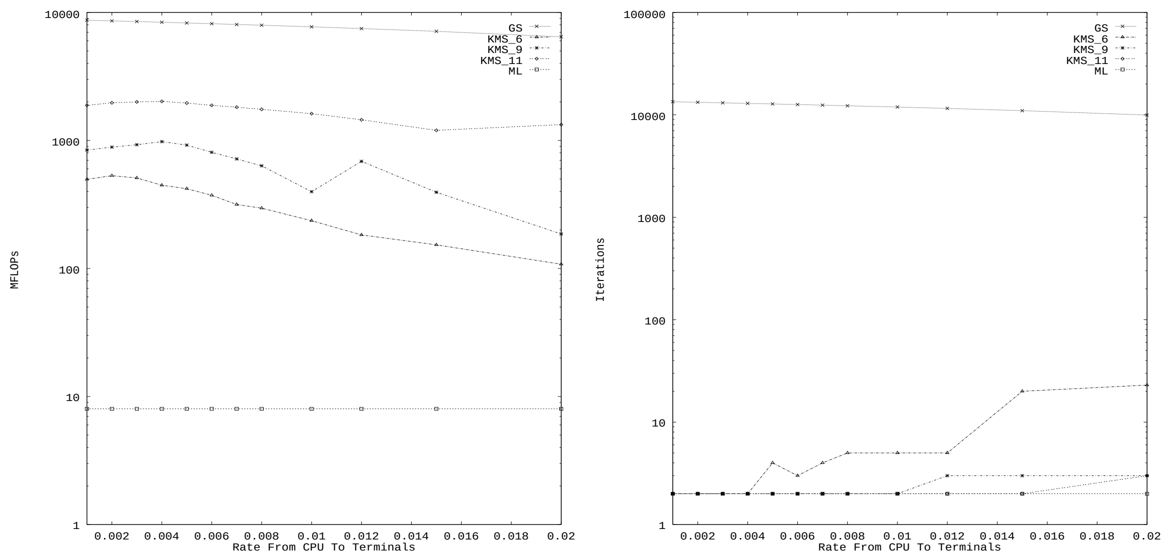

Figure 8: Interactive Computer System Model, 50 jobs, processing rate varied. Left: MFLOPs; Right: Number of Iterations.