行政院原子能委員會

委託研究計畫研究報告

多重容錯控制器研究

Investigation on Multi-Modular Redundancy Fault-Tolerant Controllers

計畫編號：1012001INER010

受委託機關(構)：中原大學機械工程學系

計畫主持人：丁鏞

聯絡電話：(03)2654319

E-mail address：yung@cycu.edu.tw

核研所聯絡人員：陳昌國

報告日期：　101 年 12 月 5 日

目　錄

中文摘要

本計畫針對使用在核能級控制器中的多重容錯控制器研究，內容包括控制器架構、容錯機制、系統可靠度分析等議題，探討多重容錯控制器執行安全功能時的容錯機制(Fault Tolerance)及可靠度(Reliability)並建立評估機制。系統在操作中，不論使用時間長短，都可能發生錯誤而故障。以監測診斷系統而言，任何一個故障的發生都可能造成該監測診斷系統執行不正常、中斷、或損壞系統本身，甚或影響受監測之系統。對一監測診斷系統之容錯機制(或容錯系統)而言，能即時發現故障，找出故障的原因，經由重組過程及容錯控制法則，使故障對監測診斷系統之影響降至最低，並使其繼續完成任務，是相當重要的。研究重點包括建立容錯系統之架構；評估各種故障可能之原因、地點、與影響；依容錯分析結果，設計平行性與分析性備份件；配合電腦輔助設備進行系統整合測試；找出最佳的備份件設計方法；採用 Markov Chain 模型並搭配類神經網路與基因演算法，針對容錯系統進行可靠度之分析與改善。

Abstract

The objective of the project is to develop a fault-tolerant system. When a system is in operation, it may encounter any failure conditions regardless of the operation time period. For a monitoring and diagnostic system, any type of fault occurrence can cause the system performance to be abnormal and interrupted, or even damages the system being monitored. Thus, for a monitoring and diagnostic system, it is essential to reduce the effects of failures and accomplish the remaining tasks. This goal can be achieved by the use of appropriate redundancies. The critical tasks include: establish the fault-tolerant structure; evaluate the causes, effects, and locations of faults; based upon the analysis, design the parallel and analytical redundancies; conduct experiments through computer-aided equipment; obtain an optimal deign for the parallel and analytical redundancies; analyze and improve the fault-tolerant system's reliability through the Markov Chain model, artificial neural network, and genetic algorithm.

壹、計畫緣起與目的

根據國際上運轉中核電廠儀控系統數位化更新及新建核電廠整廠數位儀控系統之應用與申照經驗，掌握自主關鍵技術有助於申照過程及長期運轉維護策略。因此，發展自主核能儀控產業技術，對於持續發展核能應用、確保核能安全與減少外匯損失有具體之效益。另外，透過自主型核能級儀控產業技術與關鍵組件認證平台之建立，可促進國內儀控系統與零組件產業升級，除應用於核電廠外，可擴大應用於其他工業，如火力發電、石化廠等之高可靠度安全需求相關系統。

多重容錯控制器研究的行為影響甚鉅，在安全數位儀控系統中為一相當重要之議題，本計劃針對使用在核能級控制器中的多重容錯控制器進行探討，研究內容含括控制器架構、容錯機制、系統可靠度分析等議題，探討多重容錯控制器執行安全功能時的容錯機制(Fault Tolerance)及可靠度(Reliability)並建立評估機制。

本計畫之重點工作項目包括多重容錯控制器之設計與實現、多重容錯控制器之可靠度分析、可靠度評估分析機制之建立；規劃多重容錯控制器軟硬體與系統測試及驗證程序。

本計畫之最終目標包含協助核能電廠建立核能級控制器容錯之評估機制，以作為制訂國內核能級控制器設計規範之重要基礎；依據核能級控制器設計規範，建立一套多重容錯控制器應用於核能電廠之測試與驗證程序。

本計畫亦期望將研究成果運用於人員訓練，內容包括藉由多重容錯控制器的研究，建立其制訂核能級控制器設計規範之能力；藉由多重容錯控制器測試程序的規劃，增進其整合系統安全與儀控系

統之能力；藉由應用於核能電廠的多重容錯控制器驗證程序的建立，增進其開發核能級控制器之能力。

貳、研究方法與過程

Since 1970s, fault tolerance (FT) technologies have been widely developed and applied to the areas of computers [1, 2], aerospace [3, 4], chemical engineering [5, 6] and manufacturing [7, 8], robotics [9, 10], etc. The FT technologies are essential to not only the evaluations but also the enhancements of the reliability, maintainability, and survivability of the system. Carter and Bouricius [1] reviewed the redundancy techniques to self-repair the computers and improve the system reliability.

This report focuses on the Partitioning and Parallel-processing of Markov Chain (PPMC) for a fault-tolerant system of Multi-Modular Redundant (MMR) controllers. A Markov Chain (MC) is formulated to represent the N distinct states of the MMR controllers as well as the failure and repair rates between any two states. The complex MC graph has N-square directed edges as each edge weight represents the transition rate of the start state directing to the end one. The system reliability, which is calculated in terms of the edge weights, requires large computational calculations. To this end, the techniques of partitioning MC have been studied to reduce the complexity of reliability analysis. The lightly weighted edges are suspended based on the threshold parameters and the MC is partitioned into multiple weakly connected sub-graphs using the sparse matrix partitioning.

However, the partitioned MC is still a complex multidisciplinary system that requires a considerable amount of memory for processing on a single processor. To evaluate the system reliability from the partitioned MC in a memory-efficient manner, the task-farming paradigm is adopted for parallel processing. The task-farming

paradigm has a master-workers pattern. The master is responsible for decomposing a task, distributing sub-tasks among workers, and gathering partial results from workers to coordinate the final calculation for the reliability evaluation. The worker processes execute in a simple cycle: get a sub-task, process the sub-task, and send the result to the master. In addition, due to the task-farming paradigm's dynamic load balancing characteristic, the proposed methodology can respond well to the failure of some processors, which simplifies the creation of robust applications that are capable of surviving from the loss of workers or even the master.

The efficiency and accuracy of reliability analysis depend on the suspension threshold, the partition level, and the availability of parallel processors. The worst-case reliability is evaluated to compensate for inaccuracy due to the suspended edge weights. Higher suspension thresholds produce simpler MC models and faster calculations but more conservative reliabilities. The performance of PPMC is maximized to find the optimal suspension and partition levels. The proposed methodology has been successfully integrated with the detection and diagnosis processes in the fault-tolerant system. The simulation results show that, compared with the reliability analysis of the intact Markov Chain model, the proposed methodology is capable of improving the performance but also producing allowable accuracy of the reliability analysis.

一、Reliability analysis of multi-modular redundant controllers

Suppose the MMR controllers have N distinct states $S = \{1, 2, ..., N\}$, the transitions between each state are represented

in a Markov Chain (MC). Figure 1 illustrates the graph $G(S,E)$ of the MC where the edge set E is composed of N^2 directed edges: $\{E_{1,1}, E_{1,2}, ..., E_{i,j}, ..., E_{N,N}\}$. Each directed edge has two edge weights $\mu_{i,j}$ and $Q_{i,j}$. The prior one is the transition probability of that the state *i* moves to the state *j* while the later one represents the rate of change of the transition probability. The total transition probability of that the state *i* moves to every state, including itself, equals one, namely

$$\sum_{j=1}^{N} \mu_{i,j} = 1 \text{ for all } i \tag{1}$$

Therefore, the total rate of change of probability is zero, as in

$$\sum_{j=1}^{N} Q_{i,j} = 0 \text{ for all } i \tag{2}$$

Given a vector of the initial state probabilities, $\mathbf{p}(0) = 1\mathbf{e}_1 + \sum_{i=2}^{N} 0\mathbf{e}_i$, at the time $t = 0$, the vector of the state probabilities, $\mathbf{p}(t) = \sum_{i=1}^{N} p_i(t)\mathbf{e}_i$, is desirable for the evaluation of the system reliability.

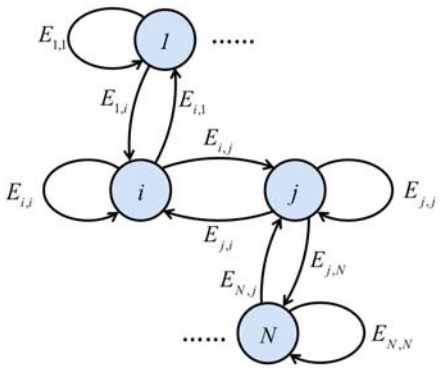


Figure 1. N-state Markov Chain for MMR controllers.

(一) Continuous-time Markov chain

To evaluate the transient behaviors of the continuous-time Markov Chain (CTMC), the differential equation of the transition probabilities is given as

$$\mathbf{p}'(t) = \mathbf{Q} \cdot \mathbf{p}(t) \tag{3}$$

In Eq. (3), $\mathbf{Q}$ is the infinitesimal generator of the CTMC [11] and it is defined as

$$\mathbf{Q} \equiv \sum_{i=1}^{N} \sum_{j=1}^{N} Q_{i,j} \mathbf{e}_j \mathbf{e}_i = \lim_{\delta \to 0} \frac{\mathbf{A}(\delta) - \mathbf{I}}{\delta} \tag{4}$$

where $\mathbf{A} \equiv \sum_{i=1}^{N} \sum_{j=1}^{N} \mu_{i,j} \mathbf{e}_j \mathbf{e}_i$ is called the transition matrix.

The differential equation (3) yields to the eigenvalue problem as in

$$(\mathbf{Q} - \lambda \mathbf{I})\mathbf{p} = \mathbf{0} \tag{5}$$

The characteristic equation is formulated by $|\mathbf{Q} - \lambda \mathbf{I}| = 0$. For the i-th eigenvalue λ_i, the eigenvector $\mathbf{v}_i = \sum_{j=1}^{N} v_{i,j} \mathbf{e}_j$ is calculated by $(\mathbf{Q} - \lambda_i \mathbf{I})\mathbf{v}_i = \mathbf{0}$. In the end, the transient state probabilities are given as

$$\mathbf{p}(t) = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_N \end{bmatrix} \begin{bmatrix} C_1 \exp(\lambda_1 t) \\ \vdots \\ C_N \exp(\lambda_N t) \end{bmatrix} \tag{6}$$

Substituting the initial conditions to Eq. (6), the coefficients $C_1 ... C_N$ can be determined by

$$\begin{bmatrix} C_1 \\ \vdots \\ C_N \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 & \square & \mathbf{v}_N \end{bmatrix}^{-1} \begin{bmatrix} p_1(0) \\ \vdots \\ p_N(0) \end{bmatrix} \tag{7}$$

where $p_1(0)=1$ and $p_i(0)=0$ for $i=2...N$.

For most MC, the time-dependent state probabilities can be evaluated using the described method. However, the computational complexity increases dramatically when the number of states increases in the MMR controllers. It is very difficult to solve the large-scale eigenvalue problem and obtain the state probabilities in allowable working time. To this end, we will focus on the discrete-time approach, which requires less calculation.

(二) Discrete-time Markov chain

With the Markov property, the state variable $X(n+1)$ at the time $n+1$ depends only upon its state at the time n, $X(n)$. That is, given the present state of the system $X(n)$, the future state of the discrete-time Markov Chain (DTMC) $X(n+1)$ is independent of its previous discrete-time states $X(0), X(1), ..., X(n-1)$. A DTMC with the finite state space S is time homogeneous if the Eq. (8) holds.

$$\mu_{i,j} \equiv \mathrm{P}\left[X(n+1)=j \,\middle|\, X(n)=i \right] = \mathrm{P}\left[X(1)=j \,\middle|\, X(0)=i \right] \tag{8}$$

for $n \geq 0$ and $i, j = 1...N$. For the DTMC, $\mu_{i,j}$ is called the one-step transition probability at the time n. In this report, we assume the one-step transition probability remains constant for all times n. The state probabilities will be calculated using the given

one-step transition probabilities.

We are interested in the probabilities $\mathrm{P}\left[X\left(n\right)=j \right]$ for all $j\in S$ and $n\geq 0$. Statistically, the probability $\mathrm{P}\left[X\left(n\right)=j \right]$ is calculated by

$$\mathrm{P}\left[X\left(n\right)=j \right]=\sum_{i=1}^{N}\mathrm{P}\left[X\left(0\right)=i \right]\mathrm{P}\left[X\left(n\right)=j\middle| X\left(0\right)=i \right] \tag{9}$$

The Eq. (9) is further rewritten in the matrix form as in

$$\mathbf{p}\left(n\right)=\mathbf{A}\left(n\right)\cdot\mathbf{p}\left(0\right) \tag{10}$$

where $\mathbf{p}\left(n\right)=\sum_{j=1}^{N}\mathrm{P}\left[X\left(n\right)=j \right]\mathbf{e}_j$ indicates the state probabilities at n; $\mathbf{p}\left(0\right)=\sum_{i=1}^{N}\mathrm{P}\left[X\left(0\right)=i \right]\mathbf{e}_i$ is for the initial probabilities; $\mathbf{A}\left(n\right)=\sum_{i=1}^{N}\sum_{j=1}^{N}\mathrm{P}\left[X\left(n\right)=j\middle| X\left(0\right)=i \right]\mathbf{e}_j\mathbf{e}_i$ is the matrix of the n-step transition probabilities. Since the DTMC is assumed to be time homogeneous, the matrix $\mathbf{A}\left(n\right)$ can be calculated by the n-the power of $\mathbf{A}$ [11], as in

$$\mathbf{p}\left(n\right)=\mathbf{A}^n\cdot\mathbf{p}\left(0\right) \tag{11}$$

To sum up, the transition probabilities of the DTMC are calculated by the matrix operations in Eq. (11). The matrix multiplication algorithm has message-passing characteristics and is more applicable in message-passing systems, such as distributed memory based cluster of computers. In the section 3, the MC is partitioned into smaller systems in order to distribute the loads to several computers. In the section 4, the parallel processing of the MC is introduced.

二、Partitioning of Markov chain

This section presents the partitioning of a large-scale MC. Essentially, the lightly weighted edges are suspended based on the threshold parameters and the MC is partitioned into multiple weakly connected sub-graphs using the sparse matrix partitioning. The created sub-graphs can be processed individually to produce partial results, which are combined to form the complete state probability distribution at a desired time instant.

(一) Reduction of Markov chain

In this report, we want to first reduce the complexity of the MC in terms of eliminating the connectivity of the weakly linked edges. Chou and Lin [12] have eliminated the edges of minimal probabilities in the MC and greatly reduced the required calculations. The evaluated network probability of the reduced Markov Chain (RMC) was slightly underestimated and can be utilized as a worst-case probability, that is, the true probability of a MC is at least higher than or equal to the worst-case probability in the RMC.

A binary dependency matrix $\mathbf{D}$ represents the connectivity of the MC, as in

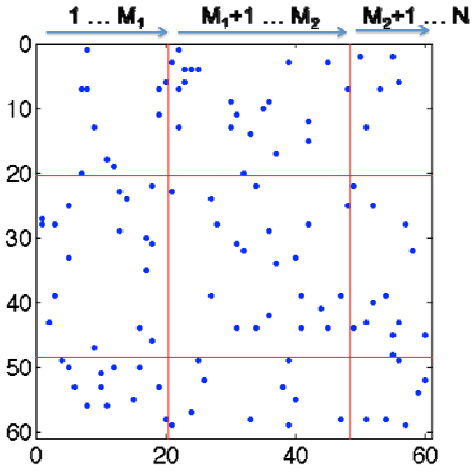
$$\mathbf{D} = \sum_{i=1}^{N} \sum_{j=1}^{N} D_{i,j} \mathbf{e}_j \mathbf{e}_i \tag{12}$$

where $D_{i,j} = 1$ when the i-th state is directed to the j-th one; otherwise, $D_{i,j} = 0$. It is noted that $D_{i,j} = D_{j,i}$ only when there exist an reversible transition between the nodes *i* and *j*; however, $\mu_{i,j}$ and $\mu_{j,i}$ do not need to be the same. Given a threshold parameter τ, the connectivity $D_{i,j}$ of the RMC is reduced to

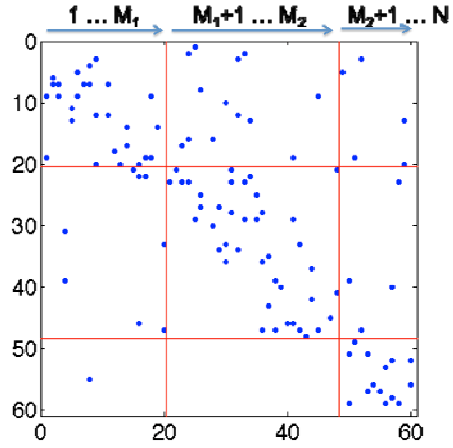
zero if $\mu_{i,j} < \tau$. As a result, the dependency matrix becomes sparse. We want to reorder the sparse dependency matrix and partition them into smaller matrices in order to distribute the computational workloads to multiple computers.

(二) M-Partitioning

Figure 2 (a) shows an example of the partition of a 60×60 dependency matrix. Each blue dot represents a non-zero element in the matrix. The four red lines divide the matrix into 9 smaller matrices, including a $M_1 \times M_1$ submatrix, a $M_1 \times \left(M_2 - M_1 \right)$ submatrix, …, and a $\left(N - M_2 \right) \times \left(N - M_2 \right)$ submatrix. Each submatrix is required for the calculations of the state probabilities in Eq. (11). Given the initial permutation of the sparse matrices $\mathbf{\Pi} = \sum_{i=1}^{N} i\mathbf{e}_i$, we focus on finding the optimal permutation to minimize the connectivity between the partitioned systems. In other words, we want to reorder the dependency matrix such that some submatrices are closer to zero matrices, as shown in Figure 2 (b). In this way, the required matrix multiplications are reduced.



(a)



(b)

Figure 2. Partitions of (a) an original and (b) a reordered dependency matrices.

Define a M -partition that partitions the dependency matrix of the RMC, $\mathbf{D}$, into two sub-systems:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_1 & \mathbf{\Phi}_1 \\ \mathbf{\Phi}_2 & \mathbf{D}_2 \end{bmatrix} \tag{13}$$

where $\mathbf{D}_1$ and $\mathbf{D}_2$ are $M \times M$ and $\left(N-M\right)\times\left(N-M\right)$ sub-matrices respectively. The sizes of the off-diagonal matrices

$\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ are $M \times \left(N-M\right)$ and $\left(N-M\right) \times M$ respectively. Meanwhile, the permutation vector is partitioned into two vectors $\mathbf{\Pi}_1 = \sum_{i=1}^{M} i\mathbf{e}_i$ and $\mathbf{\Pi}_2 = \sum_{i=M+1}^{N} i\mathbf{e}_i$. The partition parameter M is controlled to balance the computational loads for each computer. When $M < N/2$, the computational complexity of the first sub-problem is lower; on the other hand, the second sub-problem is more complex when $M > N/2$. Multiple matrix partitions can be accomplished using multiple M -partitions.

Based on the M -partition, the calculation in Eq. (11) is partitioned as in

$$\begin{bmatrix} \mathbf{p}_1\left(n\right) \\ \mathbf{p}_2\left(n\right) \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{\Psi}_1 \\ \mathbf{\Psi}_2 & \mathbf{A}_2 \end{bmatrix}^n \begin{bmatrix} \mathbf{p}_1\left(0\right) \\ \mathbf{p}_2\left(0\right) \end{bmatrix} \tag{14}$$

where $\mathbf{A}_k = \sum_{i \in \mathbf{P}_k} \sum_{j \in \mathbf{P}_k} D_{i,j}\mu_{i,j}\mathbf{e}_j\mathbf{e}_i$ is calculated based on the permutation $\mathbf{\Pi}_k$. Furthermore, the off-diagonal matrices Ψ_1 and $\mathbf{\Psi}_2$ are calculated by $\sum_{i \in \mathbf{P}_1} \sum_{j \in \mathbf{P}_2} D_{i,j}\mu_{i,j}\mathbf{e}_j\mathbf{e}_i$ and $\sum_{i \in \mathbf{P}_2} \sum_{j \in \mathbf{P}_1} D_{i,j}\mu_{i,j}\mathbf{e}_j\mathbf{e}_i$ respectively. The state probabilities in the first and second sub-problems are then determined by

$$\mathbf{p}_1\left(n\right) = \mathbf{A}_1^n\mathbf{p}_1\left(0\right) + f_1\left(\mathbf{\Psi}_1, \mathbf{\Psi}_2\right) \tag{15}$$

$$\mathbf{p}_2\left(n\right) = \mathbf{A}_2^n\mathbf{p}_2\left(0\right) + f_2\left(\mathbf{\Psi}_1, \mathbf{\Psi}_2\right) \tag{16}$$

If the permutations of the matrices and vectors are reordered such that the total probability in the off-diagonal matrices $\mathbf{\Psi}_1$ and $\mathbf{\Psi}_2$ are close to zero, the functions f_1 and f_2 in Eqs. (15) and (16) become negligible. Thus, the state probabilities can be

approximated by the following equation:

$$\mathbf{p}_i(n) \cong \mathbf{A}_i^n \mathbf{p}_i(0) \tag{17}$$

(三) Iteration process of finding the minimal coupling between sub-problems

The objective of our approach is to find the permutation order that minimizes the total transition probability between the two subsystems:

$$p_C = \sum_{i=1}^{M} \sum_{j=1}^{N-M} \Psi_{1,ij} + \sum_{i=1}^{N-M} \sum_{j=1}^{M} \Psi_{2,ij} \tag{18}$$

One of the possible methods is to diagonalize the dependency matrix $\mathbf{D}$; however, it does not ensure the coupling probability p_C is minimized for the specific M-partition. In this report, an effective iteration process is proposed as in the following steps to determine the optimal permutation order:

1. Given the initial permutation $\mathbf{\Pi}^{(0)}$ and dependency matrix $\mathbf{D}^{(0)}$. Calculate the initial coupling probability $p_C^{(0)}$ using the Eq. (18).
2. Begin with $\Psi^{(0)}{}_{1,N-M}$ in the upper-right off-diagonal dependency matrix as the pivot element. The iteration number k equals one.
3. Suppose the location of the pivot element is (i, j). If the (i, j)-th element in $\Psi_1^{(k-1)}$, or the (j, i)-th element in $\Psi_2^{(k-1)}$, is not zero, the column number of the pivot element is denoted as $c_p^{(k)}$ and go to step 5; otherwise, go to step 4.
4. If the pivot element is the last element $\Psi_{M,1}^{(k-1)}$, cannot advance

further and go to step 10. Otherwise, move to the adjacent element along the zig-zag trajectory, shown in Figure 3, and consider it as the new pivot element. Go to step 3.

5. Begin with the first column, i.e. $c = 1$. Let $\mathbf{\Pi}^{(k)} = \mathbf{\Pi}^{(k-1)}$ and $n_C^{(k)} = n_C^{(k-1)}$.
6. If $c = c_p^{(k)}$, move to the adjacent column by $c = c + 1$. If $c \le N$, go to step 7; otherwise, go to step 9.
7. Interchange c and $c_p^{(k)}$ in the permutation $\mathbf{\Pi}^{(k-1)}$ and calculate the coupling probability p_I of the interchanged off-diagonal dependency matrix.
8. If $n_I < n_C^{(k)}$, $n_C^{(k)} = n_I$ and assign the interchanged permutation as $\mathbf{\Pi}^{(k)}$. Advance to the adjacent column by $c = c + 1$ and go to step 6.
9. If $n_C^{(k)} = n_C^{(k-1)}$, the iteration process has been converged. Go to step 10. Otherwise, update the interchanged matrix $\mathbf{D}^{(k)}$ using the determined permutation $\mathbf{\Pi}^{(k)}$ and advance to the next iteration by $k = k + 1$. Go to step 4.
10. The optimal permutation is $\mathbf{\Pi}^* = \mathbf{\Pi}^{(k)}$ and the optimal reordered dependency matrix is $\mathbf{D}^* = \mathbf{D}^{(k)}$.

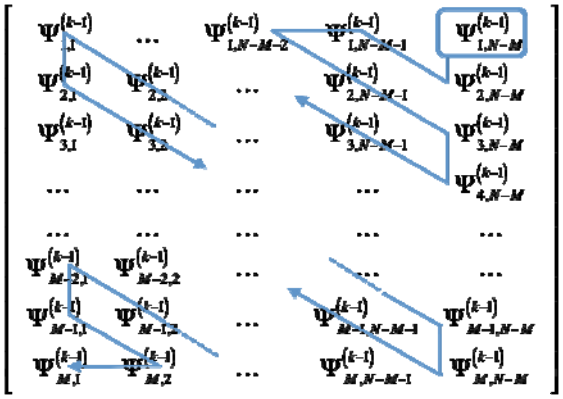


Figure 3. Determination of pivot elements along the zig-zag trajectory in the off-diagonal dependency matrix.

三、Parallel-processing of Markov chain

As illustrated in the section 3.2, the partitioned MC has multiple transition probability sub-matrices. As also mentioned in the section 3, these square matrices can be processed individually to produce partial results, which are combined to form the complete state probability distribution at a desired time instant. However, for a large-scale MC that has been partitioned, each of its transition probability sub-matrices might still need a large memory space for processing on a single processor. Thus, the task-farming paradigm is adopted for parallel processing. The task-farming paradigm has a master-workers pattern. The master is responsible for decomposing a task, distributing sub-tasks among workers, and gathering partial results from workers to coordinate the final calculation for the reliability evaluation. The worker processes execute in a simple cycle: get a sub-task, process the sub-task, and send the result to the master. In addition, due to the task-farming paradigm's dynamic load balancing characteristic, the proposed methodology can respond well to the

failure of some processors, which simplifies the creation of robust applications that are capable of surviving from the loss of workers or even the master. In this report, for simplicity purposes, with a static load balancing feature, the task-farming paradigm is used to implement an algorithm to obtain the state probability vector at any desired time instant.

The message-passing programming paradigm is one of the earliest and most widely used approaches for programming parallel computers. Its wide-spread adoption can be attributed to the fact that it imposes minimal requirements on the underlying hardware [13].

The most natural message-passing architecture for matrix operations is a two-dimensional mesh, where each node in the mesh computes one element, or submatrix, of the result array. The mesh connections allow messages to pass between adjacent nodes in the mesh simultaneously. The Cannon's algorithm [14], a memory efficient algorithm, is implemented for matrix multiplication. It uses a mesh of processors with wraparound connections to shift submatrices.

For clarity, elements, instead of submatrices, of the arrays *a* and *b* are used to illustrate the algorithm as follows.

Cannon's algorithm:

1. Initially, processor $\pi_{i,j}$ has elements $a_{i,j}$ and $b_{i,j}$ $(0 \le i < n, 0 \le j < n)$
2. Elements are moved from their initial position to an aligned position. In other words, the complete *i*-th row of *a* is left-shifted *i* positions, and then the complete *j*-th column of *b*

is up-shifted *j* positions. This step places the element $a_{i,j+i}$ and the element $b_{i+j,i}$ in processor $\pi_{i,j}$. This pair of elements are required to calculate the element $c_{i,j}$.

3. Each processor $\pi_{i,j}$ multiplies its elements.
4. The *i*-th row of *a* is shifted one position left, and the *j*-th column of *b* is shifted one position upward. This step brings together the adjacent elements of *a* and *b*, which are also required in the computation of $c_{i,j}$.
5. Each processor $\pi_{i,j}$ multiplies the elements brought to it and adds the result to the accumulating sum.
6. Repeat steps 4 and 5 until the final result of $c_{i,j}$ is obtained. In other words, given *n* rows and *n* columns of elements, a total of $n-1$ shifts need to be conducted.

Next given that both **a** and **b** have s^2 submatrices. Each submatrix has $m \times m$ elements. Thus, based on the above Cannon's algorithm, for both **a** and **b,** the initial alignment requires a maximum of $s-1$ shift operations. Afterward, there are another $s-1$ shift operations required for computation purposes. Each shift operation performs a communication involving $m \times m$ elements. Therefore, the communication time t_{comm} can be determined using Eq. (19).

$$t_{comm} = 4\left(s-1\right)\left(t_{startup} + m^2 t_{data}\right) \tag{19}$$

Thus, the Cannon's algorithm has a communication time complexity of $O(sm^2) = O(mn)$, where $s = n/m$ is assumed in

this report.

As for the computation aspect, each submatrix multiplication requires m^3 multiplications and m^3 additions. Therefore, with $s-1$ shifts as mentioned above, the computation time t_{comp} can be determined using Eq. (20).

$$t_{comp} = 2sm^3 = 2m^2 n \tag{20}$$

Hence, the Cannon's algorithm has a computation time complexity of $O(m^2 n)$.

In this report, the Cannon's algorithm is implemented through the Message Passing Interface (MPI) [15, 16]. MPI is a specification for building a library that provides standard functions for writing portable and efficient message passing programs to be run on a variety of parallel computers. In this report, the selected MPI library is a widely used open source library called MPICH2 [17, 18].

參、主要發現與結論

一、Numerical Examples

In this section, we study the Markov Chains for three different examples to validate the proposed PPMC method.

(一) Example 1: 60-state MC for MMR controllers

The first example considers a randomly generated 60-state MC, which is composed of 277 distinct directed edges. Figure 4 (a) shows the dependency matrix of the 6-state MC while each blue dot represents one of the directed connections. The blank areas stand for no connections between the states. The subfigure (a) can also be used to represent the transition matrix; in this case, each blue dot is a non-zero transition probability $\mu_{i,j}$.

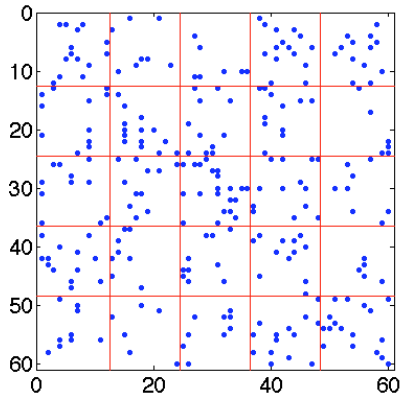
When the size of MC is in the order of tens of states, it is not necessary to reduce the MC using the threshold parameter. In the latter case when the MC size is larger, the reduction can help improve the numerical efficiency. To uniformly distribute the computational workloads to five computers, we partition the dependency matrix at the positions of $M = \{12, 24, 36, 48\}$. The red lines represent the desirable locations for matrix partitioning. According to the desirable partitioning locations, the dependency matrix is reordered such that the totally probability in the off-diagonal submatrices, shown in the gray areas in the subfigure (c), is minimized. The ratio of the total coupling probability and the total probability is 9.66 / 60 = 0.16. In other words, around 84 percent of workloads are uniformly distributed in the five diagonal submatrices. The discrete-time state probabilities are calculated using the Eq. (17).

Suppose the initial condition is given as $p_{15}(0)=1$ and $p_i(0)=0$ for $i=1...14,16...60$, the state probabilities at the time $n=10$ are calculated using the proposed method. Furthermore, the numerical performances of the PPMC method are compared with the calculations using the original MC in Eq. (10). Three different simulation results are listed in Table 1. The first case considers the second state as the fail state. The original matrix multiplication shows the failure probability is 0.07% while the proposed method overestimates the failure rate as 12.5%. A worst-case reliability, 87.50%, is then found, that is, the true probability, 99.93%, is at least larger than or equal to the underestimation, 87.50%.

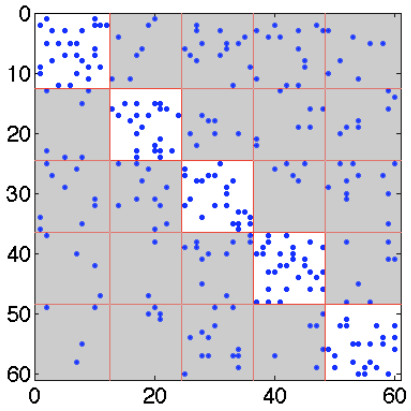
The second case shows a higher failure probability of 3.06% when the 24th state is considered as the fail state. In this case, the proposed method is capable of finding the overestimated failure probability of 12.5% and the worst-case reliability of 87.5%. In the last case, the 30th state is considered as the fail state. In this case, the PPMC method obtains the worst-case reliability of 99.87%, which is just slightly lower than the true probability of 99.88%.

Table 2 shows the comparison of computation times in this 60-state example. The average computation time for evaluating the system reliability through the original MC using one processor is 74.97 (ms). By Cannon's algorithm, the average computation time for evaluating the system reliability through the original MC using 25 processors is 3.48 (ms). The average computation time for evaluating the system reliability through the

PPMC method using five processors is 0.66 (ms). Compared with the first configuration, the second and third configurations have parallel speedup ratios of 21.53 and 113.77, respectively.



(a)



(b)

Figure 4. The (a) original and (b) partitioned dependency matrices for the 60-state Markov Chain.

Table 1. Results of the 60-state Markov Chain.

Method	Index of fail state	Probability of fail state	System reliability
Original	2	0.07%	99.93%
PPMC	2	12.5%	87.50%
Original	24	3.06%	96.94%
PPMC	24	12.5%	87.50%
Original	30	0.12%	99.88%
PPMC	30	0.13%	99.87%

Table 2. Comparison of computation times.

	Number of processors	Computation time (ms)	Speedup ratio
Original	1	74.97	1
Original	25	3.48	21.53
PPMC	5	0.66	113.77

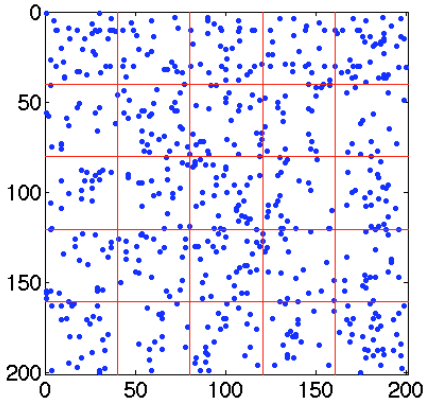
(二) Example 2: 200-state MC for MMR controllers

The second case focuses on a 200-state MC, which contains 605 directed edges. Figure 5 (a) shows the dependency matrix of the given MC. In this case, the MC reduction is also unnecessary. To distribute the workloads to five computers, the desirable partitioning locations are $M = \{40, 80, 120, 160\}$, indicated by the red lines. The off-diagonal probabilities are minimized to reorder the dependency matrix. As a result, the off-diagonal probability, the sum of probabilities in the gray areas, equals 34.33, which is around 17.16% over the total probability. The values in the off-diagonal submatrices will be neglected in Eq. (17) yielding the underestimated measure of system reliability. Therefore, the

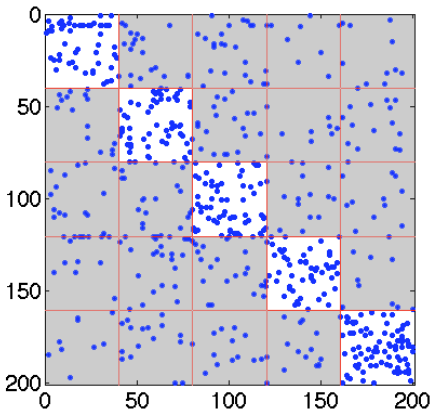
worst-case reliability can be determined.

We now want to demonstrate different types of simulation results from the first example. Suppose the 5th state is considered as the fail state, three different initial states are studied. The 1st case starts with $\mathbf{p}(0) = 1\mathbf{e}_{32}$. The original matrix multiplication shows the fail probability of 6.63% as the PPMC overestimates the fail probability of 7.05%; therefore, a worst-case reliability of 92.95% is found. Another initial condition considers $\mathbf{p}(0) = 1\mathbf{e}_{18}$. The proposed method underestimates the reliability, i.e. 82.83% < true probability = 83.25%, and utilizes the worst-case measure as a robust estimator of the system probability. Lastly, when the 99th state is considered as the initial state, a lower reliability of 77.01% is found using the original calculations. Even for the special situation of low reliability, the proposed method is capable of finding the worst-case reliability, i.e. 75.32%. Table 3 lists the detailed information about the simulation results.

Table 4 shows the comparison of computation times in this 200-state example. The average computation time for evaluating the system reliability through the original MC using one processor is 2093.91 (ms). By Cannon's algorithm, the average computation time for evaluating the system reliability through the original MC using 25 processors is 119.49 (ms). The average computation time for evaluating the system reliability through the PPMC method using five processors is 22.34 (ms). Compared with the first configuration, the second and third configurations have parallel speedup ratios of 17.52 and 93.74, respectively.



(a)



(b)

Figure 5. The (a) original and (b) partitioned dependency matrices for the 200-state Markov Chain.

Table 3. Results of the 200-state Markov Chain.

Method	Index of initial state	Probability of fail state	System reliability
Original	32	6.63%	93.37%
PPMC	32	7.05%	92.95%
Original	18	16.75%	83.25%
PPMC	18	17.17%	82.83%
Original	99	22.99%	77.01%
PPMC	99	24.68%	75.32%

Table 4. Comparison of computation times

	Number of processors	Computation time (ms)	Speedup ratio
Original	1	2093.91	1
Original	25	119.49	17.52
PPMC	5	22.34	93.74

(三) Example 3: 480-state MC for MMR controllers

The final example considers a large-scale MC that contains 480 states. Figure 6 (a) shows the dependency matrix with 1899 blue dots, that is, there are 1899 distinct directed edges in the MC. In a large problem like this, the matrix reduction can effectively improve the numerical performances. We assume the threshold parameter is $\tau = 0.1$. Figure 6 (b) shows the dependency matrix of the reduce Markov Chain, which now only contains 710 directed edges. The matrix permutation is then reordered and the resultant dependency matrix is shown in the subfigure (c). The total off-diagonal probability equals 110.37, which is around 23% of the total probability. On the other hand, 77% of the workloads are uniformly distributed to eight different computers. The red lines represent the given partitioning locations $M = \{60, 120, 180, 240, 300, 360, 420\}$.

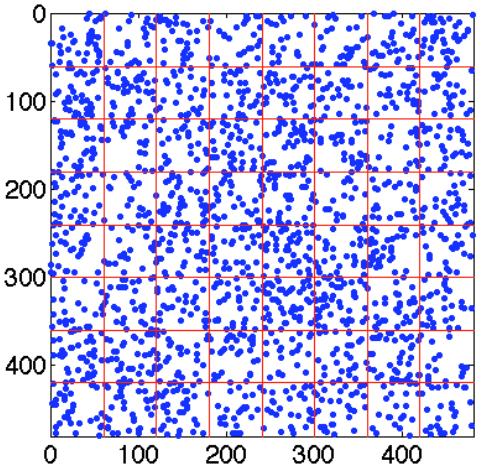
Table 5 lists two different cases of the simulations. The first case considers the 10th and 406th states as the initial and fail states. The proposed method is able to find the worst-case reliability, 99.04%, which is slightly lower than the true measure, 99.75%. In the other case of that the 432nd and 31st states are the initial and fail states, the PPMC method obtains an

underestimation of the system reliability, i.e. 70.79% << true probability = 99.90%, due to the errors from the matrix reduction and approximated calculation in Eq. (17). However, the proposed method still guarantees that the true system reliability is at least larger than or equal to the underestimated measure.

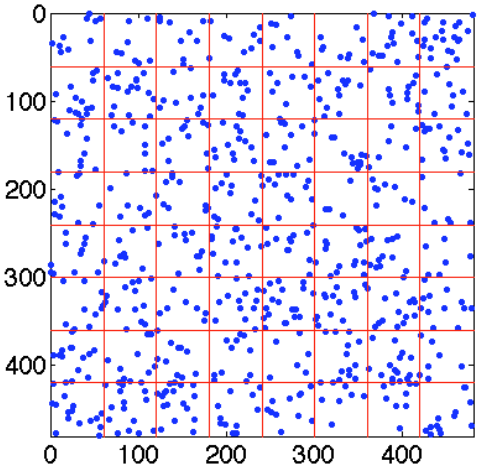
Table 6 shows the comparison of computation times in this 480-state example. The average computation time for evaluating the system reliability through the original MC using one processor is 24376.94 (ms). By Cannon's algorithm, the average computation time for evaluating the system reliability through the original MC using 16 processors is 2441.48 (ms). The average computation time for evaluating the system reliability through the PPMC method using eight processors is 75.59 (ms). Compared with the first configuration, the second and third configurations have parallel speedup ratios of 9.98 and 322.48, respectively.

Table 5. Results of the 480-state Markov Chain.

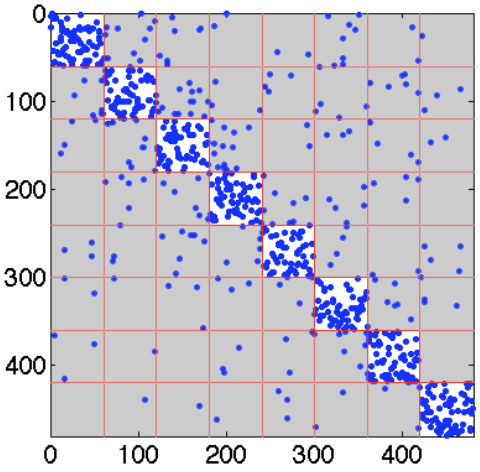
Method	Index of initial state	Index of fail state	Probability of fail state	System reliability
Original	10	406	0.25%	99.75%
PPMC	10	406	0.96%	99.04%
Original	432	31	0.10%	99.90%
PPMC	432	31	29.22%	70.79%



(a)



(b)



(c)

Figure 6. The (a) original, (b) reduced, and (c) partitioned dependency matrices for the 480-state Markov Chain.

Table 6. Comparison of computation times

	Number of processors	Computation time (ms)	Speedup ratio
Original	1	24376.94	1
Original	16	2441.48	9.98
PPMC	8	75.59	322.48

二、Conclusions

This report focuses on the reliability analysis of multi-modular controllers using discrete-time Markov chains. A novel approach is proposed to reduce and partition a large-scale Markov chain model into multiple independent sub-models, which can be tackled naturally by parallel processing in the first place. However, each of those sub-models might still require a huge amount of memory for processing on a single processor. Thus, the task-farming paradigm of parallel processing is used to implement a memory-efficient algorithm to obtain each sub-model's result, and combine the results to form the complete state probability vector at any desired time instant.

The results of three numerical examples reveal that, compared with the reliability analysis of the intact Markov Chain model, the proposed methodology is capable of improving the performance but also maintaining allowable accuracy of the reliability analysis. The amount of memory required on a single processor to perform matrix calculations is significantly reduced. The computational speed is significantly improved as well.

肆、數學符號

$\mathbf{a}$	An array in Cannon's algorithm.
$\mathbf{A}$	Transition matrix.
$\mathbf{A}_i$	i-th partitioned transition matrix.
$\mathbf{b}$	Another array in Cannon's algorithm.
c	Column number.
c_p	Column number of the pivot element in dependency matrix.
C_i	i-th coefficient.
$\mathbf{D}$	$N \times N$ dependency matrix.
$\mathbf{D}_i$	i-th partitioned dependency matrix.
D_{ij}	Connectivity parameter. D_{ij} equals one when i-th and j-th states are connected; otherwise, it is zero.
$\mathbf{e}_i$	i-th normal basis.
m	Dimensional parameter of submatrices for multiple processors.
M	Dimensional parameter for the partitioned matrix.
n	Time parameter in the computer processing.
N	Number of states in Markov Chain.
$\mathbf{p}$	Vector of transition probabilities.
p_C	Total probability of the partitioned Markov Chains couple with each other.
$\mathbf{p}_i$	i-th partitioned vector of transition probabilities.
p_I	Total coupling probability for the interchanged dependency matrix.
P(A)	Probability of A.

$P(A \mid B)$	Conditional probability of A given B.
$\mathbf{Q}$	Infinitesimal generator of continuous-time Markov Chain.
s	A dimensional parameter.
S	Finite state space.
t	Time.
$X(t)$	Variable with Markov property at time t.
λ_i	i-th eigenvalue.
$\mu_{i,j}$	Transition probability (failure/recovery rate) of the state i moves to the state j.
$\mathbf{v}_i$	i-th eigenvector.
δ	A small number.
$\pi_{i,j}$	A processor with elements $a_{i,j}$ and $b_{i,j}$.
$\mathbf{\Pi}$	$N \times 1$ permutation vector.
$\mathbf{\Pi}_i$	i-th partitioned permutation vector.
τ	Threshold parameter.
$\mathbf{\Phi}_i$	i-th off-diagonal dependency matrix.
$\mathbf{\Psi}_i$	i-th off-diagonal transition matrix.

Superscript

$'$	Derivative with respect to time.
*	Optimal solution.
(k)	N-th Iteration in finding the optimal partition.
(n)	n-th Step in computer processing.

伍、參考文獻

1.Carter, W. C., and Bouricius, W. G., 1971, "A Survey of Fault Tolerant Computer Architecture and its Evaluation". *Computer*, **4**(1), pp. 9-16.

2.Lala, J. H., 1986, "Fault Detection, Isolation, and Reconfiguration in the Fault Tolerant Multiprocessor". *Journal of Guidance, Control, and Dynamics*, **9**(5), pp. 585-592.

3.Moerder, D. D., Halyo, N., Broussard, J. R., and Caglayan, A. K., 1989, "Application of Precomputed Control Laws in a Reconfigurable Aircraft Flight Control System". *Journal of Guidance, Control, and Dynamics*, **12**(3), pp. 325-333.

4.Handelman, D. A., and Stengel, R., 1989, "Combining Expert System and Analytical Redundancy Concepts for Fault-Tolerant Flight Control". *Journal of Guidance, Control, and Dynamics*, **12**(1), pp. 39-45.

5.Hopkins, A. L., and Himmelblau, D. M., 1988, "Artificial Neural Network Models for Knowledge Representation in Chemical Engineering". *Computers & Chemical Engineering*, **12**(9-10), pp. 881-890.

6.Basila, M. R., Stefanek, G., and Cinar, A., 1990, "A model-object based supervisory expert system for fault tolerant chemical reactor control". *Computers & Chemical Engineering*, **14**(4-5), pp. 551-560.

7.Chintamaneni, P. R., Jalote, P., Shieh, Y.-B., and Tripathi, S. K., 1988, "On Fault Tolerance in Manufacturing Systems". *Network, IEEE*, **2**(3), pp. 32-39.

8.Villa, A., 1988, "A Hierarchical Knowledge-Based/Analytical Approach to Fault-Tolerant Control in Flexible Manufacturing". *IEEE International Conference on Robotics and Automation*, pp. 1120-1125.

9.Yates, S. W., and Williams, R. D., 1988, "A Fault-Tolerant

Multiprocessor Controller for Magnetic Bearings". *IEEE Micro*, pp. 6-17.

10. Chladek, J. T., 1990, "Fault Tolerance for Space Based Manipulator Mechanisms and Control Systems". In Proceeding of the First Int. Symposium On Measurement and Control in Robotics, Houston, TX.
11. Stewart, W. J., 2009, *Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling*, Princeton University Press.
12. Chou, Y.-C., and Lin, P. T., 2012, "Efficient Design Optimization of Multi-State Flow Network for Multiple Commodities". In International Conference on Mechanical, Aeronautical and Manufacturing Engineering, Tokyo, Japan.
13. Grama, A., Gupta, A., Karypis, G., and Kumar, V., 2004, *Introduction to Parallel Computing*, Pearson Education.
14. Cannon, L. E., 1969, "A cellular computer to implement the Kalman Filter Algorithm". Montana State University.
15. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., and Dongarra, J., 1998, *MPI: The Complete Reference - The MPI Core*, MIT Press, Cambridge, Massachusetts.
16. Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., and Snir, M., 1998, *MPI: The Complete Reference - The MPI-2 Extensions*, MIT Press, Cambridge, Massachusetts.
17. *MPICH2*, Argonne National Laboratory, http://www.mcs.anl.gov/research/projects/mpich2/.
18. Gropp, W., 2002, "MPICH2: A New Start for MPI Implementations". In Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag.