Vol.2015-SLDM-172 No.11
2015/10/27

DTTR方式による高信頼マルチコアシステムの性能評価に関する一考察

佐藤 謙介[1,a)]　齋藤 寛[2,b)]　米田 友洋[3,c)]　今井 雅[1,d)]

概要：我々はマルチコアシステムの高信頼化方式として，DTTR(Duplication with Temporary TMR and Reconfiguration) と呼ばれる方式を提案している．本稿では，DTTR 方式を適用したマルチコアシステムに関して，タスクグラフとシステムモデルを入力とし，全故障パターンで必要なステップ数を求め，あるミッションタイムを仮定した時のシステムの平均障害率により性能を評価する方式を提案する．また，そのツールを用いて適切なタスクコピーの属性割り当て方式を検討した結果を示す．

1. はじめに

チップ内に複数のプロセッサコアを搭載したマルチコアシステムとして，チップマルチプロセッサ (Chip Multi-Processor:CMP)，マルチプロセッサ・システムオンチップ (Multiple-Processor System-on-a-Chip)，メニコア (Many-core) システム等，様々な構成が提案されている．これらのマルチコアシステムは性能向上に用いるだけでなく，信頼性向上にも用いられている．例えば，二つのプロセッサコアを静的なペアとして同じ処理を行い，クロック毎やスレッド毎にシステム状態や演算結果の比較を行うロックステップと呼ばれる方式がある．このロックステップ方式では，比較結果が不一致であることにより故障を検出することは出来るが，故障の判定・演算結果の訂正を行うことは出来ない．そこで，ロックステップ構造を 2 つ使用して同じ処理を二つのロックステップ構造で実行することで，4 プロセッサコアのうちいずれか一つのプロセッサコアに生じた故障をマスクできるロックステップ・ペア方式が自動車業界等で用いられている．また，航空機業界等では，異なるアーキテクチャで作られた 3 つのモジュールで同一処理を行い，処理結果の多数決を取ることで単一故障をマスクすることが出来る三重化冗長 (Triple Modular Redundancy:TMR) 方式が古くから用いられている．三重化冗長方式を 3 個以上のプロセッサコアを持つマルチコアシステムに適用する場合，永久故障と同定されたプロセッサコアの使用を停止し，余っている予備のコアを用いて再構成を行い，処理を継続する方式を取ることが出来る．しかしながら，一般的にプロセッサの障害率は非常に小さいため，常に三重化冗長実行を行うのは電力面などでオーバーヘッドが大きい．

これに対して我々は，マルチコアシステムにおける高信頼化方式の一つとして，単一プロセッサ故障モデルの基で，通常時はタスクの二重実行を行って結果を比較することで故障検出を行い，結果が不一致となり故障が検出された時だけ一時的に三重化冗長実行を行い，故障が発生したプロセッサコアの同定とシステムの再構成を行う DTTR (Duplication with Temporary TMR and Reconfiguration) と呼ばれる方式を提案している [1–6]．[4] では，マルコフチェインによるモデル化を行い，上述の他の冗長方式であるロックステップペア方式及びスペア付き TMR 方式と比較し，DTTR 方式の有効性を示した．本稿では，DTTR 方式を適用したマルチコアシステムに関して，タスクグラフとシステムモデルを入力とし，全故障パターンで必要なステップ数を求め，あるミッションタイムを仮定した時のシステムの平均障害率を求める性能評価方式を提案する．また，それを用いて適切なタスク割り当て方式を検討した結果を示す．故障モデルとしては，単一プロセッサ故障を仮定し，故障の種類として永久故障 (Permanent Fault) と一過性故障 (Transient Fault) の両者を仮定する．

本稿の構成は以下の通りである．次節で DTTR 方式の概要について述べ，第 3 節で DTTR の性能評価方式につ

[1] 弘前大学
Hirosaki University, Aomori, 036-8561, Japan
[2] 会津大学
University of Aizu, Fukushima, 965-8580, Japan
[3] 国立情報学研究所
National Institute of Informatics, Tokyo, 101-8430, Japan
[a)] sato@hal.eit.hirosaki-u.ac.jp
[b)] hiroshis@u-aizu.ac.jp
[c)] yoneda@nii.ac.jp
[d)] miyabi@eit.hirosaki-u.ac.jp

いて提案する．第 4 節でタスクグラフを生成し，評価した結果について述べ，第 5 節でまとめを述べる．

2. DTTR 方式とは

2.1 ハードウェアプラットフォーム

DTTR 方式を適用するハードウェアプラットフォームは図 1 に示す NoC(Network-on-Chip) ベースのマルチコアシステムを対象とする．図において，R で示されるノードはオンチップネットワークルータを表し，コア間のデータの送受信はルータとオンチップネットワークを介したパケット通信により行われる．ここで，いくつかの仮定をおく．

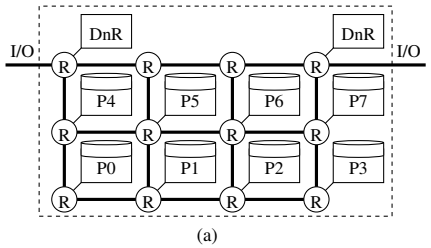

図 1 NoC モデル
Fig. 1 NoC model.

- 各プロセッサコア P0〜P7 はあらかじめ静的に複数のタスクを格納出来る容量があるローカルメモリを持つ．
- タスク実行プロセッサの選択や，演算結果の比較などの制御を行う DnR (Diagnostics and Reconfiguration) ユニットが 1 個あるいは複数ある．
- DnR とプロセッサコア間は高スループット・低レイテンシなオンチップネットワークにより接続され，オンチップネットワーク上のリンク故障・ルータ故障はディペンダブル・ルーティングアルゴリズム等 [7] によりマスクされるものとする．

2.2 アプリケーションモデル

対象とするアプリケーションはセンサからの入力に対して一定周期でアクチュエータを制御するタスクを繰り返すものとし，リアルタイム制約として一つのデッドライン制約 (タイム)CT を持つ．対象とするタスクグラフの例を図 2 に示す．

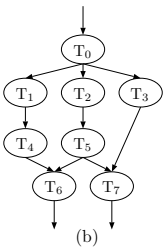

図 2 タスクグラフ例
Fig. 2 Task graph example.

図 2 はタスク数 8 のタスクグラフの例であり，センサ・アクチュエータとの送受信を行う I/O タスクは省略している．有向枝はタスク間の依存性を表す．本稿では，評価を簡単にするために，下記を仮定する．

- 各プロセッサコアにおけるタスクの実行時間は全て等しく T とする．
- デッドライン制約 CT に対し，連続した $2\times CT$ 内に故障は一つしか生じない．この制約は，内部状態の有るタスク実行において，冗長コアにおける内部状態更新を正しく行うために必要な仮定である [6]．

2.3 タスク割り当てと実行方式

与えられたハードウェアプラットフォームモデルと実行するタスクグラフに対して，DTTR 方式ではあらかじめ静的に各プロセッサコアにタスクコピーの割り当てを行い，通常時の二重化実行コア及び故障判定時の三重化実行コア，スペアのコアを設定する．これは図 3 (a) に示すタスク割り当て表を作成することにより実現される．

	P0	P1	P2	P3	P4	P5	P6	P7
T_0	A	A	S	1				
T_1					A	A	S	1
T_2	S	1	A	A				
T_3					S	1	A	A
T_4	A	A	S	1				
T_5					A	A	S	1
T_6	S	1	A	A				
T_7					S	1	A	A

(a)

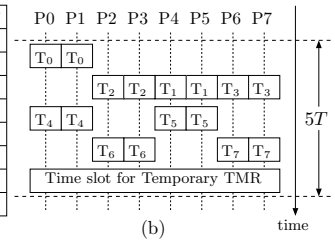

図 3 4 冗長時のタスク割り当て表と実行ステップ
Fig. 3 Task assign table for four-redundancy DTTR and execution steps.

タスク割り当て表において，A は通常時に二重実行を行うアクティブコピー (Active)，S は故障判定時に三重化実行を行うためのスタンバイコピー (Stand-by)，数値 n は永久故障が生じたときに再構成を行うためのインアクティブコピー (Inactive) である．それぞれのタスクコピーを処理するコアをアクティブコア，スタンバイコア，インアクティブコアと呼ぶ．図 3 (a) は 4 冗長の構成を示しており，インアクティブコアは 1 個のみである．初期状態のタスクの冗長数が 5 以上の場合は 2,3,$\cdots$ と設定する．

この割り当て表を基に，DnR が実行プロセッサの選択を行うことになる．DnR の動作を簡潔にまとめる．

(1) チップ外部にある制御入力 (センサ) からの入力を受ける．
(2) 割り当て表を基に，実行可能なタスクに関してタスク毎に指定された 2 つのアクティブコアに必要なデータを送る．
(3) 各プロセッサコアで演算処理が行われた後，プロセッサコアから演算結果や内部状態情報が送られてきたものを受け，ペア間で比較を行う．

(4) 比較結果が一致していれば次のタスクを行い，必要であればアクチュエータへの出力を生成する．
(5) 比較結果が一致していない場合，割り当て表を基に，当該タスクの 2 つのアクティブコアと 1 つのスタンバイコアに三重化実行に必要なデータを送る．
(6) 3 つのプロセッサコアからの演算結果や内部状態情報を受け，故障判定を行う．
- 三重化実行した結果が全て一致していれば生じた故障は一過性故障と判定し，割り当て表を更新せず次のサイクルを開始する．
- 多数決結果と異なる結果を出力したプロセッサがある場合，永久故障と判定し，同定したプロセッサコアを除いて割り当て表を更新する．S を A に，1 を S にそれぞれ更新し，インアクティブコピーは数値を 1 つずつ減らす．

(7) 冗長度が 2 以上である限り，タスク実行結果は正しいことが保証されるため，上記の動作を繰り返す．

図 2 のタスクグラフに対してタスク割り当て表 3 (a) を設定したとき，各プロセッサの実行ステップは図 3 (b) の様になる．図に示す通り，タスクグラフ全体の実行レイテンシは $4T$ であるが，故障が生じた時の故障判定のための一時的な三重化実行が必要となるため，必要なレイテンシは $5T$ となる．つまり，デッドライン制約として $4T$ が課せられると DTTR 方式は適用することが出来ず，タイミング制約は最低 $5T$ 以上が必要と言える．

3. DTTR 方式の性能評価方式

3.1 タスク割り当て表の最適化

前節で設定したタスク割り当て表において，仮にプロセッサコア P5 に永久故障が生じたとする．このとき，故障判定が正常に行われると，タスク割り当て表は表 4 (a) の様に更新される．このとき，タスクグラフ全体の実行ステップは図 4 (b) の様になり，必要なステップ数が $6T$ と増加する．そのため，デッドライン制約として $5T$ が課されている場合，P5 に永久故障が生じた時点で制約を満たすことが出来ずシステムダウンとなる．

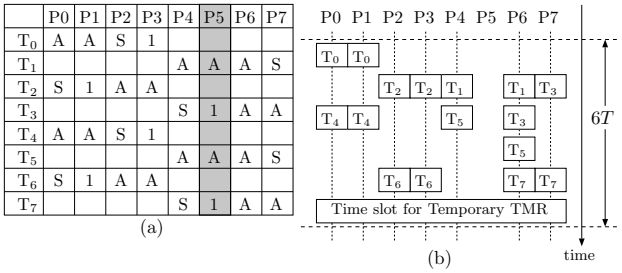

図 4　1 個故障した後の実行ステップ
Fig. 4　Execution steps after a permanent fault occurred.

一方，初期状態のタスク割り当てとして，タスク T_4〜T_7 の S と 1 の割当を入れ換えたタスク割り当て表を用いると，同じくプロセッサコア P5 に永久故障が生じた時のタスク割り当て表は図 5 (a) の様になり，タスクグラフ全体の実行ステップは図 5 (b) となるため，必要なステップ数は $5T$ のままとなる．このように，あるデッドライン制約が課された時，初期状態のタスク割り当ての違いによって，DTTR 方式で実行可能かどうかが大きく異なる．プロセッサコアの故障の組み合わせのうち，課せられたデッドライン制約内で実行可能な組合せの数が多い程，信頼度も向上すると言える．そのため，これらを反映した評価指標が必要となる．

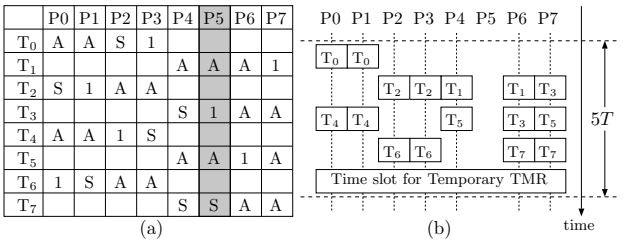

図 5　4 冗長時のタスク割り当て表 (2) と 1 個故障時の実行ステップ
Fig. 5　Task assign table (2) and execution steps after a permanent fault occurred.

3.2 システム平均障害率

一般的に，プロセッサコアの障害率を定数 λ とすると，時刻 t におけるプロセッサコア単体の信頼度は $R(t) = e^{-\lambda t}$ で表される．また，プロセッサコア単体の不信頼度 (累積故障率) は $F(t) = 1 - R(t)$ で表される．

プロセッサコア数が 8 の時，デッドライン制約を∞，初期状態の冗長度を 8 と仮定した時，システム全体の時刻 t における信頼度は以下の式で求められる．

$$R_{sys}(t) = \sum_{i=0}^{6} {}_8C_i \cdot \{R(t)\}^{8-i}\{1 - R(t)\}^i$$

上式において，i は永久故障が生じたコアの個数を表す．8 コアから成る DTTR システムで許容できる故障数の最大値は 8-2=6 個であるため，$i = 0 \sim 6$ までの各故障数での信頼度の和を取る事でシステム全体の信頼度を求める事が出来る．

ここで，あるミッションタイム τ を仮定した時，シングルコンポーネント A から成るシステムの信頼度は $R_A(\tau) = e^{-\lambda_A \tau}$ と表すことが出来る．このとき，$R_A(\tau) = R_{sys}(\tau)$ を満たすシングルコンポーネント A の障害率 λ_A をシステムの平均障害率と言い，FIT 値に換算した値は以下の式で求めることが出来る．

$$\lambda_A = -\frac{\ln\{R_{sys}(\tau)\}}{\tau} \times 10^9$$

本稿では，ミッションタイムとして $\tau = 1$ [year] =

8760 [hours] を仮定し，プロセッサコア単体の障害率 λ=100 [FIT] として平均障害率を求め，性能評価の指標とする．すなわち，平均障害率が小さければ小さい程，高信頼なシステムを実現できると言える．

さらに，タスク割り当て表の違いによる信頼度の評価を行うことを考える．前述の通り，タスク割り当て表が与えられると，故障に応じて実行ステップは一意に決まる．したがって，故障数 0 から 6 までの全ての故障パターンで必要な実行ステップ数はプログラムにより求めることが出来る．故障数が i の時の順列は ${}_8P_i$ パターンあり，その中でデッドライン制約 xT 以下で実行可能な故障パターンの数を $\#nofail_i$ とすると，デッドライン制約 xT が課せられた時の信頼度は以下の式で求めることが出来る．

$$R_{sys,xT}(t) = \sum_{i=0}^{6} \frac{\#nofail_i}{{}_8P_i} \cdot {}_8C_i \cdot \{R(t)\}^{8-i}\{1-R(t)\}^i$$

本研究では，上記のタスクグラフとシステムモデルを入力とし，全故障パターンで必要なステップ数を求め，任意のミッションタイムを課した時のシステムの平均障害率を出力するスクリプトを作成した．次節では，これを用いてタスク割り当ての最適化に関して検討した結果を述べる．

4. タスク割り当ての最適化の検討と評価結果

タスクグラフによって最適なタスク割り当ては異なる．本稿では，8 種類のタスクグラフを用いて，より良いタスク割り当てを検討する．方法は以下の通りである．

- タスクグラフの生成には tgff [8] を用いる．tgff において，各ノード (タスク) への最大入力数を 2 つ，各ノードからの最大出力数は 3 つと設定し，ランダムに生成する．生成したタスク数が 12 個，最大並列度が 3〜6 のタスクグラフ各 2 通りを図 6(a)〜(h) に示す．
- 故障数によって最適なタスク割り当ては大きく変わってくるので，本稿では故障数が少ない時により影響が出ると思われる stand-by と inactive の一番目までのタスク割り当てを重視して検討する．タスク割り当て表は表 7 の 6 通りを用いる．プロセッサコア数は 8 個 (P0〜P7) で十分なメモリ容量を持つものとし，表には書かれていないが各タスクの inactive2〜5 が空白に入る．6 つの割り当て表は A の位置を固定とし，S と 1 の割り当てをかえたものである．表 7 (a) は全て S，1 の順番に割り当て，表 7 (c) は全て 1，S の順番に割り当てる．表 7 (b)，(d) は均等に S と 1 をバラバラに割り当てた 2 通りである．表 7 (e) は前半を S，1 の順番に割り当て，後半を 1，S の順番に割り当てる．表 7 (f) は前半を 1，S の順番に割り当て，後半を S，1 の順番に割り当てた表である．
- 各タスクグラフと各タスク割り当て表を入力とし，全故障パターンの必要なステップ数を求めるツールを用いて許容できる故障パターンの数及び平均障害率を評価する．

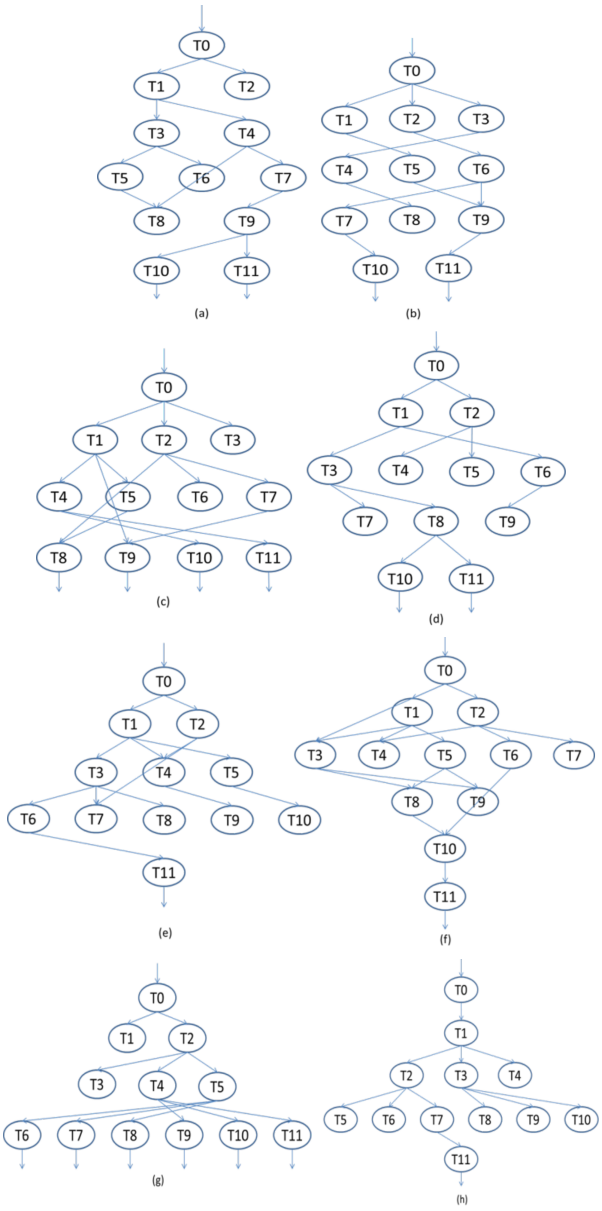

図 6 評価したタスクグラフ
Fig. 6 Evaluated taskgraphs.

各タスクグラフを 6 通りのタスク割り当てで必要なステップ数を求めた結果を表 8 に示し，平均障害率を求めた結果を表 9 に示す．表 8 の各テーブルの左側が各故障数でタスクグラフ全体を実行するのに必要なステップ数 (故障判定のための一時的な三重化実行に必要なステップ数を含まない)，右側がそのステップ数で実行可能な故障パターンが何通りあるかを示す．赤字はよい値となっている箇所を表す．表 9 の列はデッドライン制約の違いによる平均障害率の違いを表す．

表 8 から，図 6(a) のタスクグラフは各割り当て表で 3 故障まで同じ結果になっているが，その他のタスクグラフでは割り当て表 (a),(c) の結果が明確に悪いことがわかる．

1	P0	P1	P2	P3	P4	P5	P6	P7
T0	A	A	S	1				
T1					A	A	S	1
T2	S	1	A	A				
T3					S	1	A	A
T4	A	A	S	1				
T5					A	A	S	1
T6	S	1	A	A				
T7					S	1	A	A
T8	A	A	S	1				
T9					A	A	S	1
T10	S	1	A	A				
T11					S	1	A	A

(a)

2	P0	P1	P2	P3	P4	P5	P6	P7
T0	A	A	S	1				
T1					A	A	S	1
T2	1	S	A	A				
T3					1	S	A	A
T4	A	A	1	S				
T5					A	A	1	S
T6	S	1	A	A				
T7					S	1	A	A
T8	A	A	S	1				
T9					A	A	S	1
T10	1	S	A	A				
T11					1	S	A	A

(b)

3	P0	P1	P2	P3	P4	P5	P6	P7
T0	A	A	1	S				
T1					A	A	1	S
T2	1	S	A	A				
T3					1	S	A	A
T4	A	A	1	S				
T5					A	A	1	S
T6	1	S	A	A				
T7					1	S	A	A
T8	A	A	1	S				
T9					A	A	1	S
T10	1	S	A	A				
T11					1	S	A	A

(c)

4	P0	P1	P2	P3	P4	P5	P6	P7
T0	A	A	1	S				
T1					A	A	1	S
T2	S	1	A	A				
T3					S	1	A	A
T4	A	A	S	1				
T5					A	A	S	1
T6	1	S	A	A				
T7					1	S	A	A
T8	A	A	1	S				
T9					A	A	1	S
T10	S	1	A	A				
T11					S	1	A	A

(d)

5	P0	P1	P2	P3	P4	P5	P6	P7
T0	A	A	S	1				
T1					A	A	S	1
T2	S	1	A	A				
T3					S	1	A	A
T4	A	A	S	1				
T5					A	A	S	1
T6	1	S	A	A				
T7					1	S	A	A
T8	A	A	1	S				
T9					A	A	1	S
T10	1	S	A	A				
T11					1	S	A	A

(e)

6	P0	P1	P2	P3	P4	P5	P6	P7
T0	A	A	1	S				
T1					A	A	1	S
T2	1	S	A	A				
T3					1	S	A	A
T4	A	A	1	S				
T5					A	A	1	S
T6	S	1	A	A				
T7					S	1	A	A
T8	A	A	S	1				
T9					A	A	S	1
T10	S	1	A	A				
T11					S	1	A	A

(f)

図 7　タスク割り当て表
Fig. 7　Task assign tables.

3 故障までの結果を比較すると割り当て表 (b),(d) がより良い結果となっている．4 故障では比較的割り当て表 (b),(d) が良い結果が出ているが，悪い結果が出ているときもあり，割り当て表 (f) の方が良い時もあるという結果が見られる．表 9 からも割り当て表 (a),(c) の結果が悪いことがわかる．表 9 では，割り当て表 (d) が良い結果が出ていることがわかる．

実行ステップ時間が増えてしまう場合というのは，本来並列に実行できるタスクが同時に実行できなくなった場合である．割り当て表 (a),(c) のように偏ったタスク割り当てをすると，1 個故障すると active が重なってしまうため並列に実行できなくなる場合が多くなり，悪い結果が出たと思われる．割り当て表 (b),(d) のようにバラバラにタスクを割り当てた場合でも，必ずしもいい結果となるわけではなく，4 故障以降はタスクグラフによる違いが大きく結果に影響を与えている．しかし，3 故障までを想定する場合では，どのタスクグラフでも同じような傾向がみられる．今回の結果から，1 故障の結果が良ければ 3 故障まで良い結果となっていることがわかる．1 故障は 8 通りしかないので 3 故障までを想定するなら比較的簡単に良いタスク割り当てを求めることができると思われる．

図6(a)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	6	4	6	4	6	4	6	4	6	4	6	4
	7	4	7	4	7	4	7	4	7	4	7	4
2故障	6	12	6	12	6	12	6	12	6	12	6	12
	7	44	7	44	7	44	7	44	7	44	7	44
3故障	7	336	7	336	7	336	7	336	7	336	7	336
4故障	7	960	7	1056	7	960	7	1152	7	960	7	1152
	8	312	8	216	8	408	8	312	8	312	8	312
	9	312	9	216	9	216	9	216	9	312	9	120
	10	96	10	96	10	96			10	96	10	96
			11	96								

図6(b)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	6	4	6	8	6	4	6	8	6	8	6	8
	7	4			7	4						
2故障	6	12	6	44	6	12	6	44	6	44	6	44
	7	44	7	12	7	44	7	12	7	12	7	12
3故障	6	24	6	168	6	24	6	168	6	168	6	168
	7	312	7	168	7	312	7	168	7	168	7	168
4故障	7	864	7	1056	7	960	7	864	7	864	7	1056
	8	432	8	528	8	528	8	720	8	432	8	528
	9	288	9	96	9	192	9	96	9	384	9	96
	10	96										

図6(c)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	6	4	5	4	6	4	5	4	5	2	5	2
	7	4	6	4	7	4	6	4	6	6	6	6
2故障	6	12	6	44	6	12	6	44	6	44	6	44
	7	44	7	12	7	44	7	12	7	12	7	12
3故障	6	24	6	168	6	24	6	168	6	168	6	168
	7	312	7	168	7	312	7	168	7	168	7	168
4故障	7	1008	7	1296	7	1104	7	1296	7	1296	7	1296
	8	288	8	288	8	384	8	384	8	288	8	384
	9	288	9	96	9	192			9	96		
	10	96										

図6(d)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	6	4	5	2	6	4	5	2	5	2	5	2
	7	4	6	6	7	4	6	6	6	6	6	6
2故障	6	12	6	44	6	12	6	44	6	44	6	44
	7	44	7	12	7	44	7	12	7	12	7	12
3故障	6	24	6	168	6	24	6	168	6	168	6	168
	7	312	7	168	7	312	7	168	7	168	7	168
4故障	7	1008	7	1200	7	1008	7	1296	7	1104	7	1200
	8	288	8	384	8	480	8	384	8	480	8	480
	9	288	9	96	9	192			9	96		
	10	96										

図6(e)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	6	4	5	2	6	4	5	2	6	8	6	8
	7	4	6	6	7	4	6	6				
2故障	6	12	6	44	6	12	6	44	6	44	6	44
	7	44	7	12	7	44	7	12	7	12	7	12
3故障	7	336	6	144	7	336	6	144	6	144	6	144
			7	192			7	192	7	192	7	192
4故障	7	1008	7	1296	7	1104	7	1296	7	1104	7	1200
	8	288	8	288	8	384	8	384	8	480	8	384
	9	288	10	96	9	192			9	96	9	96
	10	96										

図6(f)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	7	8	6	2	7	8	6	2	6	2	6	2
			7	6			7	6	7	6	7	6
2故障	7	56	7	56	7	56	7	56	7	56	7	56
3故障	7	312	7	312	7	312	7	312	7	312	7	312
	8	24	8	24	8	24	8	24	8	24	8	24
4故障	7	888	7	1080	7	888	7	984	7	984	7	984
	8	216	8	504	8	600	8	504	8	504	8	600
	9	384	9	96	9	192	9	192	9	192	9	96
	10	192										

図6(g)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	6	4	5	2	6	4	5	2	6	8	6	8
	7	4	6	6	7	4	6	6				
2故障	6	12	6	44	6	12	6	44	6	44	6	44
	7	44	7	12	7	44	7	12	7	12	7	12
3故障	7	336	6	144	7	336	6	144	6	144	6	144
			7	192			7	192	7	192	7	192
4故障	7	984	7	1080	7	984	7	888	7	984	7	984
	8	24	8	408	8	408	8	504	8	216	8	408
	9	384	9	192	9	288	9	288	9	576	9	192
	10	288									10	96

図6(h)

	割り当て表 (a)		割り当て表 (b)		割り当て表 (c)		割り当て表 (d)		割り当て表 (e)		割り当て表 (f)	
1故障	7	8	6	8	7	8	6	8	6	6	6	6
									7	2	7	2
2故障	7	56	6	32	7	56	6	32	6	16	6	16
			7	24			7	24	7	40	7	40
3故障	7	336	7	336	7	336	7	336	7	336	7	336
4故障	7	864	7	1056	7	864	7	1248	7	1056	7	1152
	8	432	8	432	8	432	8	240	8	432	8	240
	9	192	9	192	9	384	9	192	9	192	9	288
	10	192										

図 8　4 故障までの必要ステップ数
Fig. 8　Required time steps under from one failure to four failures.

図6 (a)

デットライン制約	6+1	7+1	8+1	9+1	10+1	11+1
割り当て表 (a)	400	2.01E−06	1.14E−06	2.71E−07	2.35E−09	
割り当て表 (b)	400	1.74E−06	1.14E−06	5.38E−07	2.70E−07	1.44E−12
割り当て表 (c)	400	2.01E−06	8.73E−07	2.71E−07	2.05E−09	
割り当て表 (d)	400	1.47E−06	6.05E−07	3.22E−08		
割り当て表 (e)	400	2.01E−06	1.14E−06	2.71E−07	1.88E−09	
割り当て表 (f)	400	1.47E−06	6.05E−07	2.71E−07	2.11E−09	

図6 (b)

デットライン制約	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	400	2.28E−06	1.07E−06	2.71E−07	1.21E−09
割り当て表 (b)	0.525	1.74E−06	2.71E−07	2.93E−09	
割り当て表 (c)	400	2.01E−06	5.38E−07	2.93E−09	
割り当て表 (d)	0.525	2.28E−06	2.71E−07	2.76E−09	
割り当て表 (e)	0.525	2.28E−06	1.07E−06	2.99E−09	
割り当て表 (f)	0.525	1.74E−06	2.71E−07	2.81E−09	

図6 (c)

デットライン制約	5+1	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	800	400	1.88E−06	1.07E−06	2.71E−07	1.41E−09
割り当て表 (b)	401	0.525	1.07E−06	2.71E−07	2.93E−09	
割り当て表 (c)	800	400	1.61E−06	5.38E−07	2.93E−09	
割り当て表 (d)	401	0.525	1.07E−06	3.28E−09		
割り当て表 (e)	600	0.525	1.07E−06	2.71E−07	2.93E−09	
割り当て表 (f)	600	0.525	1.07E−06	3.28E−09		

図6 (d)

デットライン制約	5+1	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	800	400	1.88E−06	1.07E−06	2.71E−07	1.41E−09
割り当て表 (b)	600	0.525	1.34E−06	2.71E−07	2.93E−09	
割り当て表 (c)	800	400	1.88E−06	5.38E−07	2.93E−09	
割り当て表 (d)	600	0.525	1.07E−06	3.28E−09		
割り当て表 (e)	600	0.525	1.61E−06	2.71E−07	2.93E−09	
割り当て表 (f)	600	0.525	1.34E−06	3.28E−09		

図6 (e)

デットライン制約	5+1	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	800	400	1.88E−06	1.07E−06	2.71E−07	1.41E−09
割り当て表 (b)	600	0.525	1.07E−06	2.71E−07	2.93E−09	
割り当て表 (c)	800	400	1.61E−06	5.38E−07	2.93E−09	
割り当て表 (d)	600	0.525	1.07E−06	3.28E−09		
割り当て表 (e)	800	0.525	1.61E−06	2.71E−07	2.93E−09	
割り当て表 (f)	800	0.525	1.34E−06	2.71E−07	2.76E−09	

図6 (f)

デットライン制約	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	800	0.000307	1.61E−06	5.38E−07	1.76E−09
割り当て表 (b)	600	0.000307	2.71E−07	3.17E−09	
割り当て表 (c)	800	0.000307	5.38E−07	3.17E−09	
割り当て表 (d)	600	0.000307	5.38E−07	3.17E−09	
割り当て表 (e)	600	0.000307	5.38E−07	3.17E−09	
割り当て表 (f)	600	0.000307	2.71E−07	3.17E−09	

図6 (g)

デットライン制約	5+1	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	800	400	1.94E−06	1.88E−06	8.06E−07	1.99E−09
割り当て表 (b)	600	0.525	1.68E−06	5.38E−07	3.17E−09	
割り当て表 (c)	800	400	1.94E−06	8.06E−07	3.22E−09	
割り当て表 (d)	600	0.525	2.21E−06	8.06E−07	3.22E−09	
割り当て表 (e)	800	0.525	2.21E−06	1.61E−06	3.28E−09	
割り当て表 (f)	800	0.525	1.94E−06	8.06E−07	1.21E−06	1.64E−09

図6 (h)

デットライン制約	6+1	7+1	8+1	9+1	10+1
割り当て表 (a)	800	2.28E−06	1.07E−06	5.38E−07	1.64E−09
割り当て表 (b)	1.05	1.74E−06	5.38E−07	3.17E−09	
割り当て表 (c)	800	2.28E−06	1.07E−06	3.17E−09	
割り当て表 (d)	1.05	1.21E−06	5.38E−07	3.05E−09	
割り当て表 (e)	201	1.74E−06	5.38E−07	3.17E−09	
割り当て表 (f)	201	1.47E−06	8.06E−07	3.05E−09	

図 9 システム平均障害率
Fig. 9 System average failure rates.

5. まとめ

本稿では，我々が提案している DTTR 方式を適用したマルチコアシステムの性能に関して，タスクグラフとタスク割り当てを入力として全故障パターンの必要なステップ数とシステム平均障害率を求めるツールを用いて評価した結果を示した．少ない故障数ではスタンバイコピーとインアクティブコピーの 1 番目が大きく性能に影響することを示した．4 故障以降の場合のタスクグラフによる影響，またインアクティブコピー 2〜5 の割り当てによる影響などを検討するのが今後の課題である．

謝辞

本研究の一部は，科学技術振興機構 (JST) 戦略的創造研究推進事業 (CREST) によるものである．また，本研究の一部は，JSPS 科研費 15K00179, 15H02254 の助成を受けたものである．

参考文献

[1] T.Yoneda, M.Imai, N.Onizawa, A.Matsumoto, and T.Hanyu. Multi-chip NoCs for automotive applications. *Proc. of PRDC2012*, pages 105–110, 2012.
[2] Masashi Imai and Tomohiro Yoneda. Fault diagnosis and reconfiguration method for network-on-chip based multiple processor systems with restricted private memories. *IEICE Trans. Inf. & Systems*, E96-D(9):1914–1925, Sep. 2013.
[3] T.Yoneda, M.Imai, H.Saito, T.Hanyu, K.Kise, and Y.Nakamura. An noc-based evaluation platform for safety-critical automotive applications. *Proc. of APCCAS2014*, pages 679–682, 2014.
[4] 今井雅 and 米田友洋. マルチコアシステムにおける信頼度向上手法のマルコフモデルによる性能評価. **電子情報通信学会 技術研究報告** *DC2015-20*, pages 25–30, Jun. 2015.
[5] Masashi Imai and Tomohiro Yoneda. Comparing permanent and transient fault tolerance of multiple-core based dependable ecus. *Proc. CARS2015*, Sep. 2015.
[6] Tomohiro Yoneda, Masashi Imai, Hiroshi Saito, and Kenji Kise. Dependable real-time task execution scheme for a many-core platform. *Proc. DFT2015*, Oct. 2015 (to appear).
[7] Masashi Imai and Tomohiro Yoneda. Improving dependability and performance of fully asynchronous on-chip networks. *Proc. Async2011*, pages 65–76, Apr. 2011.
[8] Task graph for free. `http://ziyang.eecs.umich.edu/~dickrp/tgff/`.