Vol.2015-SLDM-172 No.12
2015/10/27

DTTR方式によるマルチコアシステムの信頼性向上のためのタスク割り当て手法の検討

齋藤　寛[1,a)]　今井　雅[2,b)]　米田友洋[3,c)]

概要：本稿では，DTTR 方式によるマルチコアシステムの信頼性向上のためのタスク割り当て手法を提案する．提案手法は，実行時間を短くしつつシステムの平均障害率を下げるために，タスクの最大並列度を基にタスクを分割し，並列に実行できるタスクを可能な限り異なる集合に入れる．次に，コア当たりの並列に実行できるタスクコピーの重なりを最小化するようタスク割り当てとタスクコピー属性の決定を行う．実験では，提案手法によって実行時間が短く，平均障害率が低いタスク割り当てとタスクコピーの決定が実現できたことを示した．

1. はじめに

最近の組込みシステムは，半導体微細化技術の向上によって複数のプロセッシングコアからなるマルチコアシステムとして実現されている．マルチコアシステムでは，複数のアプリケーションが並列に動作するため性能が高い．しかしながら，半導体微細化技術の向上は，製造ばらつきや劣化によって生じる欠陥も顕著となるため，システムの信頼性に大きな影響を及ぼす．

半導体集積回路は，主に冗長な回路を持たせることで信頼性を確保する．一般的な手法の 1 つとして，車載アプリなどで採用されている Lock-Step と呼ばれる二重化方式がある．2 つのコアで同じ処理を行った後比較を行い，結果が異なれば異常を知らせる．Lock-Step は，回路構造がシンプルで故障検出が速いのが特徴である．しかしながら，故障したコア自体を特定することができないため，異常を検出した後は動作を継続することができない．もう 1 つは，Triple Modular Redundancy (TMR) と呼ばれるコアの三重化である．3 つのコアで同じ処理を行った後にそれらの結果で多数決を行う．違う結果を出したコアを故障コアと判断し，残った 2 つのコアで動作を継続する．しかしながら，コアの三重化はコストが高い．

[1] 会津大学
University of Aizu, Aizu-Wakamatsu, Fukushima 965–8580, Japan
[2] 弘前大学
Hirosaki University, Hirosaki, Aomori 036–8560, Japan
[3] 国立情報学研究所
National Institute of Informatics, Chiyoda-ku, Tokyo 101–8430, Japan
[a)] hiroshis@u-aizu.ac.jp
[b)] miyabi@eit.hirosaki-u.ac.jp
[c)] yoneda@nii.ac.jp

マルチコアコアシステムは，多数のプロセッシングコアより構成されているので，多重実行がしやすい．著者らは [1] で，高信頼なマルチコアシステムを実現するためのプラットフォームとして，Duplication with Temporary Triple Modular Redundancy and Reconfiguration (DTTR) を提案した．DTTR では，通常はタスクを二重で実行し，結果を比較する．結果が異なる場合は，故障個所の特定のために，もう 1 つのコアを使って一時的に三重実行を行う．Lock-Step によるシステムや常時 TMR を行うシステムと比べ DTTR では，信頼性，処理時間，コストの面でバランスの良いマルチコアシステムを実現することが期待できる．

アプリケーションを DTTR で実行するためには，アプリケーションをコアにマッピングする段階であらかじめタスクのコピーを異なるコアに割り当てる必要がある．タスク割り当ては，アプリケーションの実行時間のみならず，システムの信頼性にも影響を及ぼす．著者らは [1] で，コアにタスクが割り当てられた後，故障のないパターンからシステムがダウンするまでの状態遷移をマルコフモデルで表現し，コアの平均障害率を算出することでシステムの信頼性を評価した．ここから，平均障害率を下げるためには，多くのコアが故障してもシステムがダウンしにくいタスク割り当てが必要であることが分かった．また，リアルタイムセーフティクリティカルシステムでは時間制約が与えられるので，アプリケーションの実行時間を短くするようなタスク割り当てが求められる．

本稿では，DTTR 方式によるマルチコアシステムの信頼性向上のためのタスク割り当て手法を検討する．本稿では，最初に多くのコアが故障してもシステムがダウンしにくいタスク割り当てを示す．次に，リアルタイムセーフティクリティカルシステムを想定し，実行時間を短くすることを優先させつつ、多くのコアが故障してもシステムがダウン

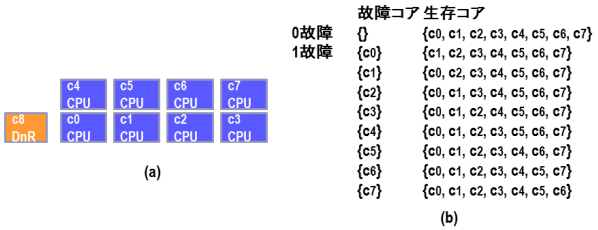

図 1　対象とするマルチコアシステム

しにくいタスク割り当てを提案する．

本稿の構成は以下のとおりである．2 節では DTTR の解説を行う．3 節では信頼性向上のためのタスク割り当て手法を述べ、4 節では実験結果を述べる．最後に，5 節ではまとめと今後の課題を述べる．

2. DTTR

DTTR は、高信頼なマルチコアシステムを実現するためプラットフォームである [1].

2.1 準備

DTTR を実現するマルチコアシステムは，図 1(a) のように複数のプロセッサコア $c_0 \ldots c_7$ と 1 つ以上の DnR (Diagnostics and Reconfiguration) コア c_8 から構成される．プロセッサコアは主にアプリケーションのタスクを実行し，DnR コアは I/O 処理，DTTR のための故障診断や後述するタスクコピーの属性の再構成を管理する．各コアは分散メモリを持つが，システム全体で共有メモリはないと仮定する．

コアは，製造時の欠陥，ソフトエラー，動作による経年劣化などにて故障する可能性がある．DTTR では，故障の原因や故障の数は重要ではないが，故障によりコアが使用できるかそうでないかの判断が必要である．故障は，ずっと使用することができない永久故障とある時間だけ誤動作となるがやがて回復する一過性故障に分類することができるが，DTTR はどちらにも対応することができる．コアの故障は，システムを構成する全てのコアにて起こり得る可能性がある．そのため，コアの故障に応じた故障パターンを定義する．故障パターンは，故障したコアの集合，および生存するコアの集合から構成される．図 1(b) は，上から故障コアがないパターン，任意の 1 コアが故障したパターンを表す．なお，ここでは DnR の故障を考えない．ある故障パターンにて，生存したコアだけで全てのタスクを二重に実行することができる場合，その故障パターンを実行可能と呼ぶ．

次に，タスクグラフとタスクコピーの属性を解説する．本稿では，アプリケーションをタスクグラフで表す．図 2(a) は，タスクグラフを表す．タスクグラフのノードはタスク t_i を表し，有向エッジはタスク間の依存関係を表す．DTTR を実現するために，アプリケーションを各コアのメモリに実装する段階で，指定された冗長度（コピー

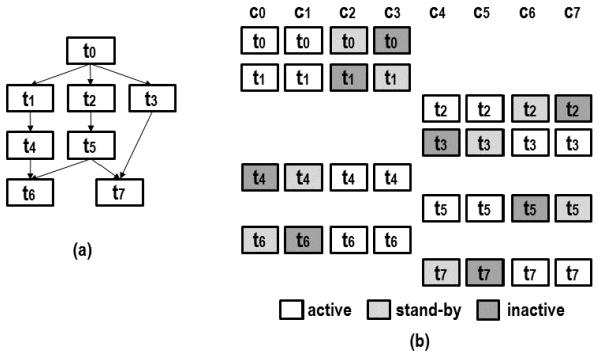

図 2　(a) タスクグラフと (b) タスク割り当て例とタスクコピーの属性

数）r の分，タスクを異なるプロセッサコアに割り当てる必要がある．また，それぞれのタスクのコピーには属性を与える必要がある．属性は，active，stand-by，inactive の 3 つである．active コピーは実際にタスクを実行するコピーである．stand-by コピーは，2 つの active コピーのうちのいずれかが故障した際にタスクを実行するコピーである．inactive コピーは，active でも stand-by でもないコピーである．故障コア無しのパターンでタスクコピーの属性を決めれば，あるコアが故障したパターンでのタスクコピーの属性が定まる．例えば，故障コア無しのパターンで active コピーの入ったコアが故障した場合，故障コア無しのパターンで stand-by コピーだったものがこの故障パターンでの active コピーとなり，inactive コピーのうちいずれか 1 つが stand-by コピーとなる．なお，$r = 2$ の場合，stand-by コピーは存在せず，$r = 3$ の場合，inactive コピーは存在しない．図 2(b) は，図 1(a) に示されたコアに対するタスク割り当てとタスクコピーの属性の例を表す．この例では，各タスクのコピー数 r は 4 である．

2.2 DTTR の動作

DTTR の動作を図 3 を用いて解説する．ここでは，図 2(a) のタスクを実行する．また，タスク割り当てとタスクコピーの属性は，図 2(b) で示されたものを用いる．なお，各タスクは 1 タイムスロット (ts) で実行されるものとする．$period_j$ は，アプリケーションの 1 回の実行時間を表す．

DTTR では初期的に，故障コアのないパターンから動作が始まる．まず，DnR コア c_8 で入力データを受け取る．c_8 は，入力信号を受信し，そのデータをタスク t_0 が active，stand-by コピーとして含まれるコア c_0，c_1，c_2 に転送する．DTTR では，通常各タスクとも 2 つの active コピーが実行される．そのため，タスク t_0 はコア c_0 と c_1 を使って実行される．stand-by コピーは故障があった時に一時的に三重実行が必要となるため，active コピー同様データを送っておくが，三重実行でない限り実行されない．各 active コピーの計算結果や状態変数は，DnR である c_8 に集約され，結果が比較される．状態変数は，アプリケーションの状態を表す変数である．active コピーの入ったコアの場合，

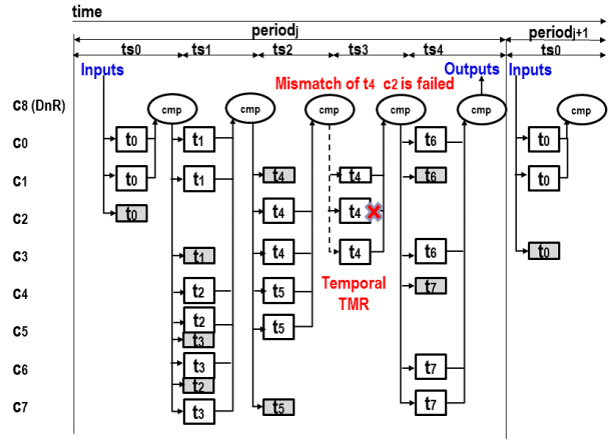

図 3　DTTR の動作

タスクの実行の前にこの変数をコアの中で保存しておく．stand-by コピーの入ったコアの場合，入力変数と共に状態変数を受け取る．そうすることで，故障があった時に状態変数が更新されたとしても，故障前の状態変数の値にロールバックすることができ，故障後もアプリケーションを正しく動作させることができる．

タイムスロット ts_2 でタスク t_4 を実行し比較した際，結果が一致しなかった場合を解説する．この場合，コア c_1 に入ったタスク t_4 の stand-by コピーを使って次のタイムスロット ts_3 で三重実行と比較 (=多数決) を行い，DnR である c_8 はどのコアが故障しているのかを特定する．DnR である c_8 はコア c_2 が故障したと判断して，今後はコア c_2 を使わずにタスクを実行するようタスクコピーの属性を再構成する．c_1 の t_4 が active コピーとなり，同様に c_2 に他のタスクの active コピーが入っている場合，他のコアに入った stand-by コピーが active コピーとなる．なお，[1] では，1 $period$ に 1 コアしか故障しないと仮定している．そのため，$period_j$ の長さは，タスクの実行タイムスロット数+1 以上と考える．+1 は TMR を行うタイムスロットを表す．一過性故障の場合，三重実行のタイムスロットで故障から回復することが期待でき結果が全て同じになる．この場合，DnR である c_8 は今後もコア c_2 を使っていくことになる．

2.3　信頼性評価

本稿では，信頼性評価に平均障害率 λ [FIT] を用いる．一般的に，リアルタイムセーフティクリティカルシステムは動作し続ける時間が要求されるため，要求された時間までの間に障害 (システムダウン) が起こる確率を信頼性評価に用いることが多い．この時間をここではミッション時間 MT と呼ぶ．マルチコアシステム sys の時間 t における信頼度を $R_{sys}(t)$，MT に対する平均障害率を λ_{sys} とすると，$R_{sys}(t)$ と λ_{sys} の関係は，以下の式で表すことができる．

$$R_{sys}(MT) = e^{-\lambda_{sys} MT} \tag{1}$$

1FIT は，$1,000,000,000$ (10^9) 時間に 1 回の故障を表す．この関係と (1) 式より，平均障害率 λ_{sys} は，以下の式で表すことができる．

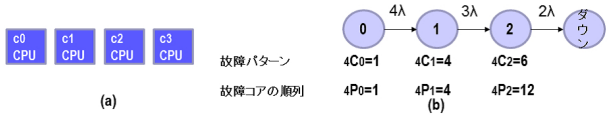

図 4　(a)4 コアのマルチコアシステムと (b) マルコフモデル

$$\lambda_{sys} = -\frac{\ln R_{sys}(MT)}{MT} * 10^9 \quad [FIT] \tag{2}$$

λ_{sys} の値が小さいほうが信頼性が高いということを意味する．

次に，システムの信頼度 $R_{sys}(MT)$ の計算式を図 4 を用いて解説する．図 4(a) は，4 コアによるマルチコアシステムで，このシステムで DTTR を実現することを考える．図 4(b) は，このシステムで故障コアのない初期状態からシステムがダウンするまでの状態遷移を表したマルコフモデルを表す．マルコフモデルのノードは状態を表し，故障コアの数を表す．有向エッジは，状態遷移を表し，生存コアのうちいずれか 1 つが故障することをコアの障害率 λ と共に表す．時間 (=状態)t における 1 コア当たりの信頼度は $R(t)$ は，以下の式で表すことができる．

$$R(t) = e^{-\lambda t} \tag{3}$$

このシステムがダウンする条件は，4 コアのうち 3 コアが故障したときである．ダウンではないが，システムの時間制約 TC として x タイムスロットが与えられると，TC 以下で全てのタスクが実行される必要がある．しかし，故障コアが増えると $period_j$ が，TC 以内にならない可能性がある．コアの故障数を i としたときに，実行時間が TC 以下となる実行可能な故障パターンに対する故障コアの順列数を $\#efps_{i,TC}$ と表す．故障コアの順列を考える理由は，故障コアのないパターンから時間を追う毎にコアが故障していくためである．0 コア故障時の信頼度は，全てのコアが生存しているということで $\{R(t)\}^4$ となる．1 コア故障時は，4 コアのうちの 1 コアが故障するので，故障パターンの組み合わせは ${}_4C_1$ となる．3 コアが生存しているということで $\{R(t)\}^3$，1 コアが故障しているということで $\{1-R(t)\}$，これらを合わせると ${}_4C_1\{R(t)\}^3\{1-R(t)\}$ となる．また，4 コアのうち 1 コアが故障した時の故障パターンに対する故障コアの順列 ${}_4P_1$ に対して，$\#efps_{1,TC}$ が実行可能な故障パターン順列数とすると，1 コア故障時の信頼度は，$\frac{\#efps_{1,TC}}{{}_4P_1}{}_4C_1\{R(t)\}^3\{1-R(t)\}$ となる．同様に，2 コア故障時の信頼度は，$\frac{\#efps_{2,TC}}{{}_4P_2}{}_4C_2\{R(t)\}^2\{1-R(t)\}^2$ となる．システム全体が n コアで故障コア数を i とすると，時間制約を TC としたときのマルチコアシステム sys の信頼度 $R_{sys,TC}(t)$ は，i コア故障までの信頼度の和より，

$$R_{sys,TC}(t) = \sum_{i=0}^{n}\left(\frac{\#efps_{i,TC}}{{}_nP_i}{}_nC_i\{R(t)\}^{n-i}\{1-R(t)\}^i\right) \tag{4}$$

となる．信頼度 $R_{sys,TC}(t)$ を高めるためには，実行可能な故障パターンに対する故障コアの順列数 $\#efps_{i,TC}$，あるいは $\#efps_{i,TC}$ を決める要因となる実行可能な故障パターン数を増やすことが求められる．

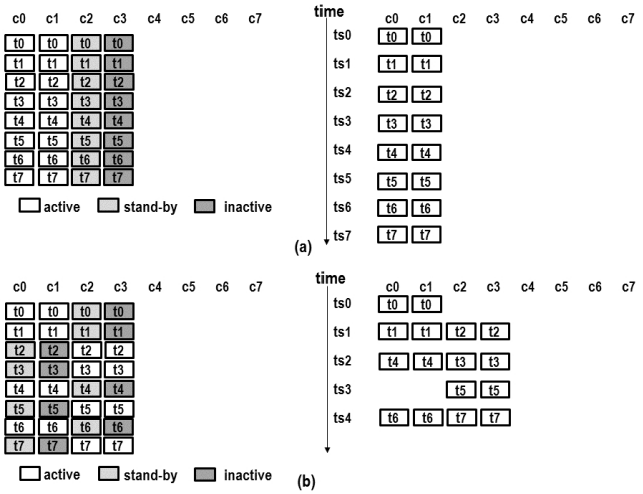

図 5 (a) タスクのコピー数 r に相当するコアのそれぞれに全てのタスクを割り当てた場合と (b) 並列実行できるタスクのコピーが同一コアに割り当てられた時，可能な限り active コピーを重ねないとした場合

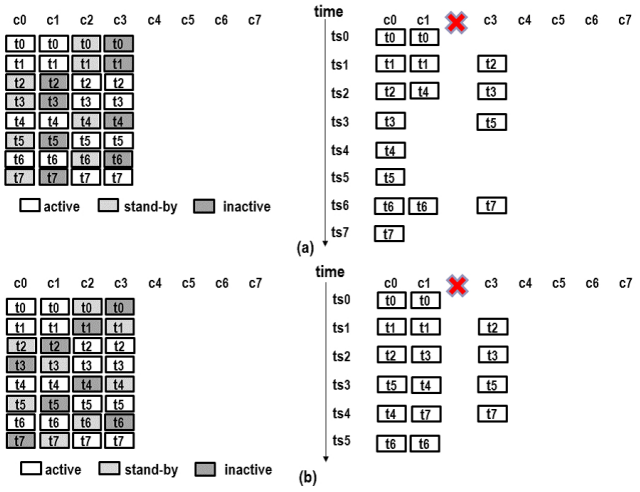

図 6 (a) 図 5(b) のタスク割り当てで，コア c_2 が故障した場合と (b) 図 5(b) のタスク割り当てで，並列実行できるタスクのコピーの stand-by コピーも可能な限り重ねないとした場合

3. 性能と信頼性を考慮したタスク割り当て手法

3.1 実行可能な故障パターンに対する故障コア順列数を最大化する割り当て

実行可能な故障パターンに対する故障コアの順列数 $\#efps_{i,TC}$ を最大化するは，タスクのコピー数 r に相当するコアのそれぞれに全てのタスクを割り当てることである．これを，図 1(a) のマルチコアシステムに図 2(a) のタスクグラフを割り当てることを例に説明する．なおここでは，DnR への割り当ては考慮せず，時間制約 TC は無限であると仮定する．また，全てのタスクは 1 ts で実行するものとする．

あるタスクに対して 4 つのコピーを割り当てる場合，DTTR を実行できない故障パターンは ${}_4C_3$ 存在する．DTTR では，タスクは通常二重実行なので，4 つのコピーが入った 4 つのコアのうち，3 つのコアが故障してしまうと DTTR を実行できない．3 つのコアの故障パターンを故障コアの順列で考えると 3! となり，${}_4C_3 * 3!$ 順列が DTTR を実行できないことになる．8 つのコアのうち 3 つのコアが故障した時の実行可能な故障パターンに対する故障コアの順列数 $\#efps_{3,TC}$ は，$\#efps_{3,TC} = {}_8P_3 - 4 * 3!$ となる．$tnum$ 個のタスクがそれぞれ異なるコアに割り当てられると仮定した場合，DTTR を実現できない故障パターンは $tnum *_4 C_3$ 個となる．DTTR を実現できない故障パターンに対する故障コア順列数の最小化は，$tnum = 1$ の場合が最も期待できる．これは，図 5(a) のようにタスクのコピー数 r に相当するコアのそれぞれに全てのタスクを割り当てたときに相当する．なお，ここではコア毎にタスクコピーの属性を合わせている．

しかしながら，$tnum = 1$ の場合，使用するコアの数が r に限定されるため，図 5(a) 右のようにタスクの並列実行が制限される．結果的にマルチコアシステムを利用しているにも関わらず性能が出ない．時間制約 TC に任意の値を与えた場合，$\#efps_{i,TC}$ が制限される．

3.2 提案手法

提案手法は，実行時間を短くしつつ実行可能な故障パターン (組み合わせ) 数の最大化を目的にタスク割り当てとタスクコピーの属性の決定を行う．順列数 $\#efps_{i,TC}$ の最大化を考えない理由は，ある故障パターンにおける実行時間は，時間制約の下，タスクスケジューリングを行わない限り分からないためである．しかしながら，3.1 節の例を見た通り，$\#efps_{i,TC}$ は実行できない故障パターンを少なくすればよい．従って，提案手法は実行可能な故障パターンの最大化を目的とする．なお，提案手法は，DnR の故障を考慮しない．また，コアあたりのタスク数も制限しない．全てのタスクは 1 ts で実行するものとし，コア間通信時間も考慮しない．

始めに，タスクコピーの属性の決定法を検討する．図 5(a) のようにタスクコピーの属性をコア毎に合わせてしまうと，並列に実行できるタスクが並列に実行できなくなる．図 2(a) より，タスク t_1，t_2，t_3 が並列に実行できるが，どのタスクもコア c_0 と c_1 に割り当てられたコピーが active なので，図 5(a) 右のように順次実行となってしまう．性能を改善するために，タスクが並列に実行できる場合，図 5(b) のように active コピーを同一コアに重ねないようにする．この場合，t_1 の active コピーはコア c_0 と c_1 を，t_2 の active コピーはコア c_2 と c_3 となる．タスク t_3 もタスク t_1 と t_2 に並列に実行できるが，タスク t_3 は次のタスク t_7 の実行まで 1 タイムスロットの余裕があるため，タスク t_1 と t_2 の実行を優先させる．結果として，図 5(b) は図 5(a) に対して, 3 ts 短くなる．

同様に，並列実行できるタスクが同一コアに割り当てられた場合，図 6(b) のように stand-by コピーも同一コアに

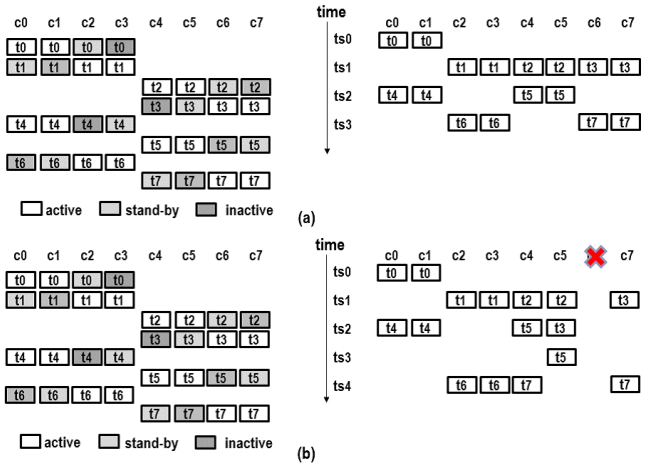

図 7 (a) タスクの並列性を考慮してタスクを 2 つの集合に分け，タスク割り当てとタスクコピーの属性を決めた場合，(b) コア c_6 が故障した場合

重ねないようにする．図 6(a) のように，並列に実行できるタスク t_2 と t_3 の stand-by コピーを同一のコア c_0 とした場合，コア c_2 か c_3 のいずれかが故障した場合，どちらも stand-by コピーが入ったコア c_0 を使おうとするので，タイムスロットが余分に必要となる．図 6(b) の割り当てでは，タスク t_2 と t_3 の stand-by コピーの入ったコアが異なるため，コア c_2 が故障しても，タスク t_2 はコア c_0 をタスク t_3 はコア c_1 を使うので，並列に実行することができる．

次に，タスクの並列性を考慮したタスク割り当て手法を検討する．3.1 節で，$tnum = 1$ の時，実行可能な故障パターンに対する故障コア順列数 $\#efps_{i,TC}$ の最大化が期待できることを示したが，時間制約 TC によってはタスクの並列実行を制限するため性能が出ず，$\#efps_{i,TC}$ が制限されることを示した．$tnum$ の数を増やすことで，性能を改善することができる．$tnum$ の数を増やすということは，いいかえるとタスクグラフのタスクを $tnum$ 個の集合に分けることに相当する．例えば，$tnum = 2$ とすると，図 7(a) のようなタスク割り当てが可能となる．このタスク割り当ては，並列に実行できるタスクを可能な限り異なる集合に入るように分けている．なお，ここでは先に述べたタスクコピー属性の決定を考慮している．この割り当ての場合，8 コア全てを使うことになるので，故障コアがないパターンでは並列度が最大となるタスク t_1，t_2，t_3 がタイムスロット ts_1 で並列に二重実行することができる．しかし，コア c_6 か c_7 のいずれかが故障した場合，タスク t_3 はコア c_5 を使うことになるため，タスク t_5 と競合が起こり，タイムスロットを延ばすことに繋がる．

以上の検討より，提案手法は以下のようにタスク割り当てとタスクコピー属性の決定を行う．

(1) タスクの最大並列度を基にタスクの集合を作り，並列に実行できるタスクをできるだけ異なる集合に入れる．
(2) コア当たりの並列に実行できるタスクコピーの重なり $overlap$ を最小化する．

前者が実現困難な場合は，時間制約に対するタイムスロットの余裕を求め，タイムスロットの余裕のあるタスクとないタスク同士を優先的に同じ集合に入れる．後者に関しては，以下の式より $overlap$ を求め，$overlap$ が小さくなるようにタスク割り当てとタスクコピーの属性の決定を行う．

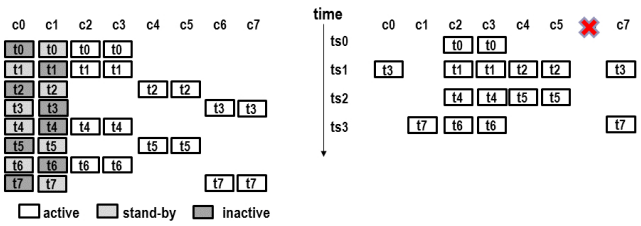

図 8 タスクの並列性を考慮してタスクを 3 つの集合に分け，タスク割り当てとタスクコピーの属性を決めた場合

$$\begin{aligned} overlap = {} & num_{a,a} * w_0 + num_{a,s} * w_1 \\ & + num_{a,i} * w_2 + num_{s,s} * w_3 \\ & + num_{s,i} * w_4 + num_{i,i} * w_5 \end{aligned} \quad (5)$$

ここで，$num_{a,a}$，$num_{a,s}$，$num_{a,i}$，$num_{s,s}$，$num_{s,i}$，$num_{i,i}$ はそれぞれ，同一コアに割り当てられた，並列に実行できるタスクの active コピー同士，active コピーと stand-by コピー，active コピーと inactive コピー，stand-by コピー同士，stand-by コピーと inactive コピー，inactive コピー同士の数を表す．w_0 から w_5 は重みを表し，$w_0 > \ldots > w_5$ とする．図 8 はそのような割り当ての 1 例である．この場合，1 コアの故障パターンの全てが，故障コア無しのパターンと同じタイムスロット数になる．

タスクグラフの最大並列度以上にタスクの集合を作った場合，更なる性能の改善（この場合，2 コア以上の故障の時の性能改善）が期待できるかもしれない．しかし，タスクの集合を分ければ分けるほど，実行可能な故障パターンが減少することになるので，$efps_{i,TC}$ も減少することとなり，信頼度が悪くなる恐れがある．また，タスクグラフの最大並列度を基にタスクの集合を作ったとしても，タスク割り当てとタスクコピーの属性の組み合わせは多数あるので，提案手法は最適性を保証するものではない．そのため今後は，シミュレーテッドアニーリングなどによって準最適解を求め，提案手法との比較を行うことで提案手法の有効性を示す．また，DTTR を想定したタスクスケジューリング手法も必要となるが，こちらは故障パターン毎にタスクスケジューリングを行う [2] をベースとする予定である．

4. 実験結果

実験では，提案手法を用いてタスク割り当てとタスクコピー属性の決定を行い，時間制約 TC が変化した時の平均障害率，および実行可能な故障パターンに対する故障コア順列数 $\#efps_{i,TC}$ を評価する．

マルチコアシステムは図 1(a) のマルチコアシステムを用い，タスクグラフは図 2(a) のタスクグラフ tg_0 と図 9 のタスクグラフ tg_1 を用いる．tg_0 の最大並列度は 3，tg_1 の

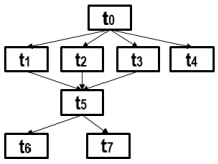

図 9　タスクグラフ tg_1

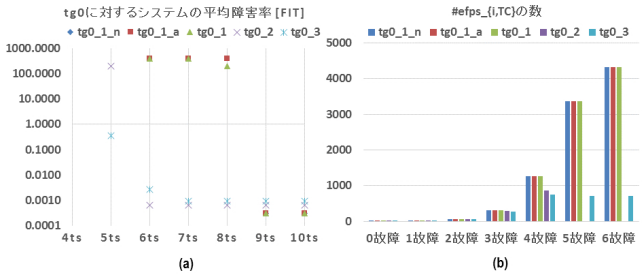

図 10　タスクグラフ tg_0 を図 1(a) のマルチコアシステムに割り当てたときの (a) 平均障害率と (b)$\#efps_{i,TC}$ の数

最大並列度は 4 である．$overlap$ 計算のために必要な重み $w_0, \ldots w_5$ はそれぞれ，6，5，4，3，2，1 とした．タスクのコピー数 r は全て 4 とする．提案手法によるタスク割り当てとタスクコピー属性の決定は，現在のところ手作業だが，システムの平均障害率 λ_{sys} の計算は，[1] で利用された Perl スクリプトを用いる．この Perl スクリプトは，タスク割り当てとタスクコピー属性を与えると，故障パターン毎にタスクスケジューリングを行い実行時間を求め，λ_{sys} と $\#efps_{i,TC}$ を求める．なお，λ_{sys} を求める際にミッション時間 MT とコアあたりの障害率 λ を与える必要があるが，$8,760$ 時間 (1 年に相当) と 10^{-7} とする．

図 10(a) は，タスクグラフ tg_0 を図 1(a) のマルチコアシステムに割り当てたときの平均障害率を表す．$tg0\_1\_n$，$tg0\_1\_a$，$tg0\_1$，$tg0\_2$，$tg0\_3$ はそれぞれ，図 5(a)，図 5(b)，図 6(b)，図 7(a)，図 8 のタスク割り当てとタスクコピー属性の決定を表す．図 10(a) は，ts の値が変化した時の平均障害率を表す．一方，図 10(b) は，$tg0\_1\_n$，$tg0\_1\_a$，$tg0\_1$，$tg0\_2$，$tg0\_3$ のタスク割り当てとタスクコピー属性の決定に対して，故障コア数毎の $\#efps_{i,TC}$ を表す．同様に，図 11(a) は，タスクグラフ tg_1 を図 1(a) のマルチコアシステムに割り当てたときの平均障害率を，図 11(b) は，故障コア数毎の $\#efps_{i,TC}$ を表す．$tg1\_1\_n$，$tg1\_1$，$tg1\_2$，$tg1\_3$，$tg1\_4$ はそれぞれ，タスクを 1 つの集合とし，r 個のコアに全てのタスクを割り当てたがタスクコピーの属性はコア毎に固定したもの，タスクを 1 つの集合とし，r 個のコアに全てのタスクを割り当てたが並列に実行できるタスクの active コピーと stand-by コピーを異なるコアにしたもの，並列に実行できるタスクを基にタスクを 2 つ，3 つ，4 つの集合にわけ，コア当たりの並列に実行できるタスクコピーの重なり $overlap$ を最小化したものを表す．なお，$tg0\_3$ と $tg1\_4$ が提案手法によって得られたものである．

図 10(a) と図 11(a) より，tg_0 も tg_1 のどちらの場合も，必要となる ts の数が少なくかつ少ない ts の時に平均障害率が低いのは提案手法によるものだということが分かる．tg_0 の場合，全ての 1 コア故障のパターンで，実行時間が故障コア無しのパターンと一致したことが原因である．tg_1 の場合，2 コア故障のパターンの多くで，実行時間が故障コア無しのパターンと一致したことが原因である．一方，図 10(b) と図 11(b) より，タスクの集合を作れば作るほど $\#efps_{i,TC}$ が減少していくのが分かる．2 コア故障による $\#efps_{i,TC}$ が全て一致しているは，タスクのコピー数を 4 としたためである．タスクのコピー数が 4 の場合，全ての 2 コア故障のパターンが実行可能となり，その時の順列数 (この場合 ${}_8P_2$) が $\#efps_{i,TC}$ となる．また，提案手法によって得られた $tg0\_3$ と $tg1\_4$ は共に，$tg0\_2$ と $tg1\_3$ より $\#efps_{i,TC}$ の総和が多い．$tg0\_2$ と $tg1\_3$ は，3 コア故障と 4 コア故障の時の $\#efps_{i,TC}$ は $tg0\_3$ と $tg1\_4$ より良いが，8 つのタスクを 4 つずつ 2 分しているため，5 コア故障と 6 コア故障に対応することができなかったのが原因である．

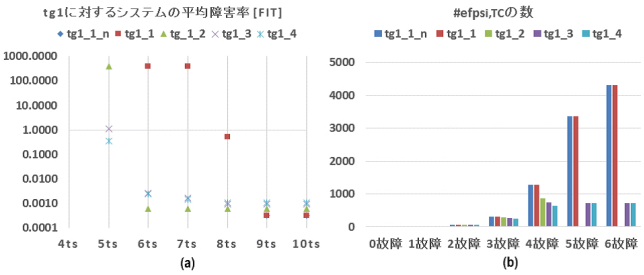

図 11　タスクグラフ tg_1 を図 1(a) のマルチコアシステムに割り当てたときの (a) 平均障害率と (b)$\#efps_{i,TC}$ の数

5. まとめ

本稿では，DTTR によるマルチコアシステムの信頼性向上のためのタスク割り当て手法を提案した．提案手法は，タスクの最大並列度を基にタスクの集合を作り，並列実行できるタスクは異なる集合に分け，コア当たりの並列に実行できるタスクコピーの重なりを最小化するようタスク割り当てとタスクコピーの決定を行う．実験から，提案手法は実行時間を短くしつつ，平均障害率が低いタスク割り当てとタスクコピーの決定が実現できることを示した．

今後はシミュレーテッドアニーリングなどによって準最適解を求め，提案手法との比較を行うことで提案手法の有効性を示す．

謝辞　本研究は JSPS 科研費 15H02254 の助成を受けたものである．

参考文献

[1] T. Yoneda, et al., ”Dependable Real-Time Task Execution Scheme for a Many-Core Platform”, *Proc. DFTS*, 2015. (to apper)

[2] H.Saito, et al., A Redundant Task Allocation Method for Reliable Network-on-Chips, *Proc. SASIMI*, 2015.

[3] K.Vallerio, *Task Graphs for Free*, 2008.