社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－135 (13)
2008／5／9

リアルタイム制約とSEU脆弱性制約の下でのヘテロジーニアスマルチプロセッサ合成技術

杉原 真[†,††]

† 豊橋技術科学大学，〒441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1
†† 独立行政法人科学技術振興機構, CREST, 〒102-0075 東京都千代田区三番町 5 番地
E-mail: †sugihara@ics.tut.ac.jp

あらまし 短期間で，かつ低コストに組込みシステムを開発する設計パラダイムとして，ヘテロジーニアスマルチプロセッサがしばしば用いられる．一方で，微細加工技術が進展するにつれて，シングルイベントアップセット (SEU: Single Event Upset) といった信頼性に関する問題が関心事となっている．本稿では，一定の処理を行う間に生じる SEU 数を SEU 脆弱性と定義し，SEU 脆弱性とリアルタイム性を考慮したヘテロジーニアスマルチプロセッサ合成技術に関する議論を行う．ヘテロジーニアスマルチプロセッサ合成問題を混合整数計画問題として定式化する．計算機実験を行い，合成されるヘテロジーニアスマルチプロセッサのリアルタイム制約，SEU 脆弱性，及びチップ面積に関する考察を行う．

キーワード SEU，ソフトエラー，リアルタイム制約，SEU 脆弱性，ヘテロジーニアスマルチプロセッサ

On Synthesizing a Heterogeneous Multiprocessor System under Real-Time and SEU Vulnerability Constraints

Makoto SUGIHARA[†,††]

† Toyohashi University of Technology, 1-1 Hibarigaoka, Tenpakucho, Toyohashi, Aichi 441-8580 Japan
†† Japan Science and Technology Agency, CREST, 5 Sanbancho, Chiyoda-ku, Tokyo 102-0075 Japan
E-mail: †sugihara@ics.tut.ac.jp

Abstract Utilizing a heterogeneous multiprocessor system has become a popular design paradigm to build an embedded system at a cheap cost within short development time. A reliability issue, which is vulnerability to single event upsets (SEUs), has not been taken into account in a conventional design flow, while chip area, performance, and power consumption have been. This paper presents a system design paradigm in which a heterogeneous multiprocessor system is synthesized and its chip area is minimized under real-time and SEU vulnerability constraints. We build an MIP model for minimizing chip area of a heterogeneous multiprocessor system under the constraints. Experimental results show that our design paradigm have achieved automatic generation of cost-competitive and reliable heterogeneous multiprocessor systems.

Key words SEU, Soft Error, Real-Time, SEU Vulnerability, Heterogeneous Multiprocessor

1. はじめに

シングルイベントアップセット (SEU: Single Event Upset) は，高エネルギーを持った粒子が半導体デバイスの脆弱な部分に衝突することで生じる信号値の反転現象である．IC 部品の SEU によって，コンピュータシステムはしばしば誤った動作，すなわちソフトエラーを引き起こす．ソフトエラー率 (SER: Soft Error Rate) はデバイスやシステムがソフトエラーを生じる頻度であり，言い換えれば，SER とは一定時間当りのソフトエラー発生数である．SER はしばしば IC 部品の脆弱性尺度として用いられる．経済基盤，交通基盤，及び製造基盤を担う電子機器においてソフトエラーを生じると，その経済損失は計り知れない．個々のアプリケーションで要求される信頼性に応じて，適切な製造コストでそれを満足するシステム設計技術の確立が重要である．

システム設計の立場からは，設計の早期にコンピュータシステムの脆弱な部分に対処するために，正確な信頼性見積もり技術，及び高信頼化設計技術の二つが重要性を増している．本稿では，一定の計算を行う間に生じる SEU 数を SEU 脆弱性を定義し，信頼性指標の一つとして SEU 脆弱性を考慮する．以下に示す理由から，各々の部品の信頼性を独立に評価するよりも，むしろコンピュータシステム全体の信頼性を評価する技術が，信頼性評価及び高信頼化設計において本質的であると考える．

（1） コンピュータシステムは CPU，SRAM，DRAM，および ASIC といった様々な部品から構成されており，各々の IC 部品は全く異なる信頼性をあわせ持つ．

（2） メモリモジュールに限定しても，パリティに代表される符号化による手法，及び，TMR(Triple Modular Redundancy) に代表される構造的手法といった様々な高信頼化設計手法がある．システム全体に対して要求される信頼性を満たすように，各部品における高信頼化設計手法を適切に選択し，不要なコスト増，性能低下，及び電力増加を避ける必要がある．

（3） コンピュータシステムの振る舞いは，ハードウェア，ソフトウェア，およびシステムに対する入力によって決定される．例えば，プログラムが変わると，コンピュータシステムの振る舞いも変わる．あるプログラムは多くのメモリ空間を要求

するかも知れないし，他のプログラムではそうでないかも知れない．他の例としては，マルチプロセッサシステムにおいて，あるプログラムは効率よく多くの CPU コアを用いるかも知れないし，他のプログラムは一つの CPU だけを用いて実行するかも知れない．使用するハードウェア資源が異なれば，SEU 脆弱性も異なると考えるのが自然である．

これまでに，いくつかのソフトエラー脆弱性見積もり技術が提案されている [1–3,7–9,13,20–24]．我々はこれまでに命令セットシミュレーション (ISS: Instruction Set Simulation) レベルでコンピュータシステムの SEU 脆弱性見積もり手法を提案している [12,13,18]．この SEU 脆弱性見積もり技術は，サイクル精度のシミュレーションを行うことにより，プログラムの実行においてメモリモジュールが如何に使用されるかを空間的かつ時間的に調べる手法である．ISS ベースの SEU 脆弱性見積もり手法の抽象度は回路シミュレーションのそれよりも高いために，高速に実行でき，またシステム設計の初期に実施することができる．ISS ベース SEU 脆弱性見積もりは短い開発期間にコンピュータシステムの脆弱な部分を特定する上で適したアプローチである．

高信頼化設計 (DFR: Design for Reliability) もまた重要な技術の一つである．パリティ，ハミング符号，および TMR はメモリモジュールにおける誤りを検出し，訂正する上で良く知られている設計技術である．Elakkumanan らは，スキャンフリップフロップを用いた論理回路の DFR 技術を提案している [4]．これは，二つのフリップフロップから構成されるスキャンフリップフロップの構造に着目し，組合せ回路の出力を二回サンプルし，それらの値を比較する時間的冗長性を用いる手法である．我々はこれまでに，コンピュータシステムのキャッシュメモリに着目し，性能と信頼性の間にはトレードオフがあることを報告している [14]．この知見を元に，我々は，SEU に脆弱なキャッシュメモリのサイズを変更できるキャッシュアーキテクチャを提案している [15]．提案するキャッシュアーキテクチャを採用したコンピュータシステムの SEU 脆弱性を最小化するタスクスケジューリング技術をシングル CPU 向け，マルチ CPU 向けに提案している [15–17,19]．SEU 脆弱性を削減するためのタスクスケジューリングにおいては，タスクの実行開始時刻，タスクが処理される CPU，及びキャッシュメモリの動作モードの変更時刻が最適に決定され，SEU 脆弱性が最小化される．このアプローチの長所は，IC が設計，製造されたあとでも，開発する要求に合わせて，システム設計者が自分のシステムの信頼性を決定できる点である．このアプローチの短所としては，キャッシュメモリ・サイズの変更を実現するための回路規模の増加とそれによる性能低下，キャッシュの動作モードの変更に伴うキャッシュメモリの内容の追い出しに生じる実行時間の増加が挙げられる．

本稿では，リアルタイム制約と SEU 脆弱性制約の下でのヘテロジーニアスマルチプロセッサ合成手法を議論する．ヘテロジーニアスマルチプロセッサ合成は，次の理由でセルベース設計を置き換えるものに成り得る．

（1） セルベース設計よりも高い抽象度で設計することは組込みシステムの開発期間を削減することができる．

（2） ASIC はそれ自身がヘテロジーニアスマルチプロセッサシステムの一プロセッサと成り得るために，ヘテロジーニアスマルチプロセッサ合成において，過去の設計資産である ASIC をシームレスに用いることができる．

我々の提案手法は，性能に関する制約を考慮するとともに，微細化の進展につれて深刻になっている SEU 脆弱性を考慮するものである．

2. ハードウェア構成による性能及び信頼性

本節では，DRAM と SRAM における SEU 脆弱性のトレンドを述べるとともに，チップ上のメモリのチップ面積占有比率のトレンドについて述べる．これらの二つのトレンドを調査することによって，システム全体の SEU 脆弱性において SRAM 回路の SEU 脆弱性が今後より支配的になるであろうことが理解される．

我々は SRAM 回路の SEU 脆弱性がコンピュータシステム全体において支配的であろうと考えている．まずはじめに，SEU 脆弱性密度，すなわち単位面積当りのソフトエラー率に着目する．SRAM の SEU 脆弱性密度は増加している一方，DRAM の SEU 脆弱性密度はほぼ横ばいである．図 1 は SRAM と DRAM の SEU 脆弱性密度のトレンドを示している．なお，このトレンドを算出する上で，ITRS によって提供されている単位面積当りのトランジスタ数のデータ [28] と，Slayman によって示されたビット当りのソフトエラー率 (CUR: Cell Upset Rate) を用いた [11]．この図は SRAM は DRAM よりも SEU 脆弱性密度が高いことを示しており，また，DRAM の SEU 脆弱性密度はほぼ横ばいである一方，SRAM の SEU 脆弱性密度は増加していくことを示している．

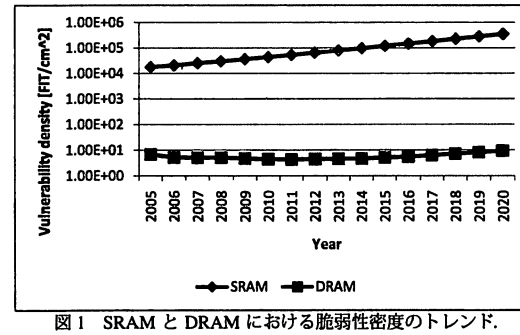

図 1 SRAM と DRAM における脆弱性密度のトレンド．

SRAM の SEU 脆弱性がより致命的になっていく次の理由として，マイクロプロセッサの性能を図るために，今後より大容量のキャッシュメモリがチップ上に搭載されるという点があげられる．図 2 は，マイクロプロセッサの性能とデータ転送能力におけるトレンドを示す．技術トレンドの観点から，二つの仮定の下に図を作成した．一つめの仮定は，マイクロプロセッサの性能はチップ上のトランジスタ数とオンチップ動作周波数の積に比例しているというものである．もう一つの仮定は，データ転送能力は信号線入出力パッド数とオフチップ動作周波数の積に比例しているというものである．図に示すように，プロセッサの性能向上に対してデータ転送能力の向上が追い付かないという状況が生じる．この乖離の埋め合わせを行うために，今後，より大容量のキャッシュメモリが用いられることは容易に想像できる．ITRS では今後，SRAM のチップ面積占有率が増加すること予想しており，逆に論理回路のチップ面積占有率が減少することを予想している [27]．たとえば，LOP (Low Operation

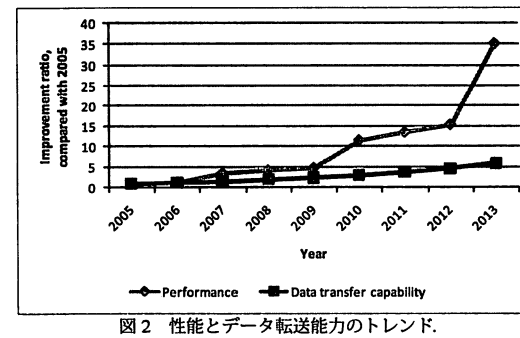

図 2 性能とデータ転送能力のトレンド．

Power) デバイスにおいては，SRAM のチップ面積占有比率が 2003 年に約 65%であり，2018 年には約 95%まで増加すると予測をしている．SEU に対して脆弱な SRAM 回路がチップ面積の大部分を占有し，その面積比率は大きくなることが予測されている．

Shivakumar らは，近い将来組合せ回路のソフトエラー率が SRAM のソフトエラー率に匹敵するほど増加し，また，逆転するという予測を発表している [10]．我々は，この予測は組合せ回路に対して多少悲観的であると考えている．彼らの計算においては，技術が進んでも SRAM のチップ占有率は一定であるという仮定を行っている．この仮定は，全段落で示した ITRS の予測，すなわち，論理回路とメモリ回路のチップ面積占有率のトレンドを反映していない．さらに，彼らの計算では，インバータチェーンのソフトエラー率を計測しており，論理的にマスクされるエラーについては考慮されていない．具体的には，Shivakumar らは，6 つのインバータの直接接続を用いて，論理回路のソフトエラー率見積もっており，この論理回路のソフトエラー率が SRAM のソフトエラー率に追い付くと予測している．以上を理由として，論理回路のソフトエラー率が SRAM のソフトエラー率に追い付く時期は，彼らが予測した 2010 年よりも後であると考えられる．システムの信頼性において，論理回路のソフトエラーが深刻になれば，論理回路の信頼性を向上するために，TMR や Elakkumanan らの手法 [4] が用いられるようになるであろう．何れにせよ，高信頼化技術の導入によって性能やチップ面積に影響が及ぶことが考えられる．

コンピュータシステムの SEU 脆弱性を精度高く見積もるためには，コンピュータシステムの動的な振る舞いを考慮する必要がある．ハードウェア，ソフトウェア，及び，コンピュータシステムに対する入力がコンピュータシステムの振る舞いを決定し，SRAM と DRAM を如何に用いるかを決定する．SRAM と DRAM の空間的かつ時間的な使用によって，プログラムの実行中に生じる SEU 数が決定される．図 3 はプログラムの実行時間と SEU 脆弱性を示すものである．ここで，プログラムの SEU 脆弱性とは，プログラムの実行中に生じる SEU 数と定義する．図より，システムの SEU 脆弱性のほとんどは SRAM に起因することが理解される．SEU 脆弱性は L1 キャッシュメモリと主記憶の SEU 脆弱性から構成されている．このデータを算出する上で，我々は ARM 社の CPU コア (ARMv4T, 200MHz) に準拠するコンピュータシステムを想定し，ベンチマークプログラムとして Mibench ベンチマークスイートのプログラムである *susan* を用いた．SRAM および DRAM のいずれにも ECC 回路があると仮定した．図 3 から，キャッシュサイズが増加すれば実行時間は減少するが，SEU 脆弱性は増加することが窺える．図より，性能と信頼性の間にトレードオフの関係があることがわかる．

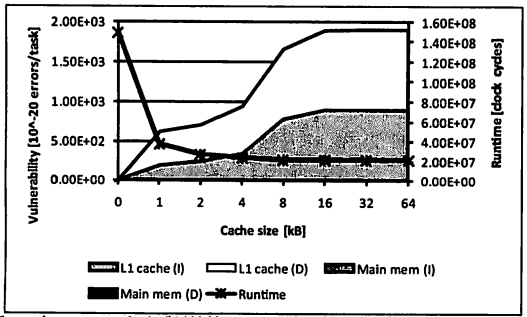

図 3　キャッシュサイズが性能と SEU 脆弱性に及ぼす影響 (input_small, smooth).

3. ヘテロジーニアスマルチプロセッサ合成

前節で議論したように，一定の処理を行う場合でも，プロセッサ毎に性能と SEU 脆弱性は異なる．前節では，SRAM の大きさに着目し，SEU 脆弱性を定量的に調査したが，メモリや論理回路に対して多重化を施し，エラーの検出・訂正を行う場合でも SEU 脆弱性が異なってくる．プロセッサのハードウェア構成によってチップ面積，性能，及び SEU 脆弱性は異なり，これらの項目を考慮できる設計方法論があることが望ましい．

この節では，ヘテロジーニアスマルチプロセッサ合成手法について議論する．本合成手法では，リアルタイム制約及び SEU 脆弱性制約の下でチップ面積が最小化される．図 4 に本合成技術における設計フローを示す．この設計フローでは，設計者はシステムの仕様を決定することから始める．仕様が決まれば，ハードウェア及びソフトウェアが開発される．ソフトウェアの開発はタスクの粒度で行うことを想定している．ハードウェアの開発においては，様々なパラメータを変化させることにより，複数のプロセッサ構成をすることを想定している．変化させるパラメータとしては，キャッシュサイズ，命令セット，ALU の数などが考えられる．全ての構成のハードウェアに対して命令セットシミュレーションを行い，実行時間及び SEU 脆弱性を計算する．これまでに提案されている脆弱性見積もり技術を用いれば，容易に SEU 脆弱性を得ることができる [1–3,7–9,12,13,18]．タスクの到着時刻及びデッドライン時刻といったリアルタイム制約はシステムの仕様によって決定される．設計者がハードウェア及びソフトウェア設計で決定した仕様，及びシミュレーション結果を用いて，ヘテロジーニアスマルチプロセッサを合成する混合整数計画 (MIP: mixed integer programming) モデルを生成する．MIP モデルに対する解を求めることにより，チップ面積が最小となるシステム構成が求められる．本節では，ヘテロジーニアスマルチプロセッサを合成する MIP モデルについて議論する．

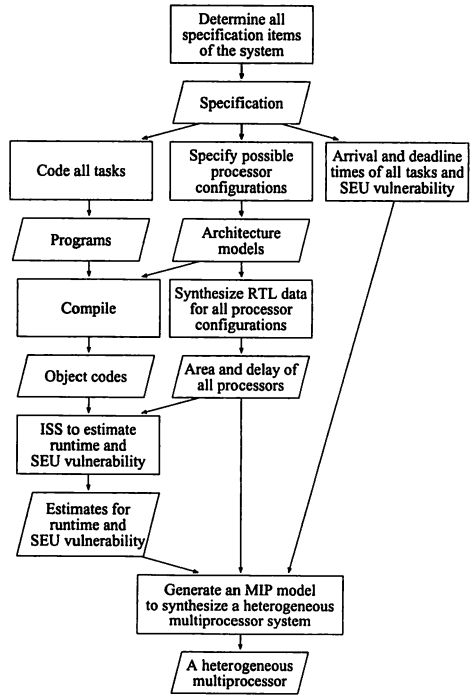

図 4　提案するヘテロ MP 合成手法における設計フロー．

MIP を作成する前に，MIP モデルについて簡単に述べる．一般に，MIP モデルは以下のように記述することができる [25]．

Minimize: $\mathbf{Ax} + \mathbf{By}$
subject to: $\mathbf{Cx} + \mathbf{Dy} \leqq \mathbf{E}$, such that $\mathbf{x} \geqq 0, \mathbf{y} \geqq 0$

ここで，**A** と **B** はコストベクトル，**C** と **D** 制約行列，**E** は定

数の列ベクトル，$\mathbf{x}$ は整数変数のベクトル，$\mathbf{y}$ は実数変数のベクトルである．効率の良い MIP ソルバが存在し，このような MIP 問題を解くことができる．

リアルタイム制約と SEU 脆弱性制約の下でチップ面積が最小であるヘテロジーニアスマルチプロセッサを合成する数理計画問題について議論する．ここで，システム上では N_{task} 個のタスクが動作し，全てのタスクはノンプリエンプティブであるとする．プリエンプションは平均的な振る舞いと最悪実行時間を要する振る舞いとの間に大きな乖離を招く．最悪実行時間と平均的な振る舞いの実行時間が近づくという意味で，ノンプリエンプションは実行時間見積もりに高い予測性を与える．タスク i $(1 \leq \forall i \leq N_{\text{task}})$ は時刻 T_{arrival_i} に到着し，実行可能となり，また，デッドライン時刻 T_{deadline_i} までに終了しなければならない．プロセッサ j 上でのタスク i の実行時間は $T_{\text{runtime}_{i,j}}$ であるとし，プロセッサ構成 k でのタスク i の SEU 脆弱性 $V_{i,k}$ とする．

本節で取り扱うヘテロジーニアスマルチプロセッサ合成問題においては，プロセッサ集合，タスクの開始時刻 $s_1, s_2, \cdots, s_{N_{\text{task}}}$，及び，タスクのプロセッサの割り当てを最適に決定し，ヘテロジーニアスマルチプロセッサのチップ面積を最小にする．ヘテロジーニアスマルチプロセッサ合成問題は以下のように定義される．

- $\mathcal{P}_{\mathcal{HMS}}$: N_{task} 個のタスク，N_{CPU} 個のプロセッサ構成，プロセッサ構成 j のチップ面積 A_j，タスク i の到着時刻 T_{arrival_i} 及びデッドライン時刻 T_{deadline_i}，及びタスク i のプロセッサ構成 k 上で実行時間 $T_{\text{runtime}_{i,k}}$ 及び SEU 脆弱性 $V_{i,k}$ が与えられたとき，(1) 全てのタスクが実行可能であり，(2) 全てのタスクがその到着時刻以降に実行を開始し，デッドライン時刻までに実行を終了し，(3) ヘテロジーニアスマルチプロセッサのチップ面積が最小になるように，ヘテロジーニアスマルチプロセッサを構成する最適なプロセッサ集合，各タスクの最適なプロセッサへの割り当て，及び各タスクの最適な実行開始時刻を求めよ．

問題 $\mathcal{P}_{\mathcal{HMS}}$ の MIP モデルの作成を行う．プロセッサ数の上限値はタスク数 N_{task} となる．変数 $x_{i,j}$ $(1 \leq i \leq N_{\text{task}}, 1 \leq j \leq N_{\text{task}})$ を以下のように定義する．

$$x_{i,j} = \begin{cases} 1 & \text{タスク } i \text{ がプロセッサ } j \text{ に割り当てられる場合．} \\ 0 & \text{その他の場合．} \end{cases}$$

また，変数 $y_{j,k}$ $(1 \leq j \leq N_{\text{task}}, 1 \leq k \leq N_{\text{CPU}})$ を以下のように定義する．

$$y_{j,k} = \begin{cases} 1 & \text{プロセッサ } j \text{ のプロセッサ構成がプロセッサ構成 } k \text{ の場合．} \\ 0 & \text{その他の場合．} \end{cases}$$

ヘテロジーニアスマルチプロセッサのチップ面積はシステムで用いられている全てのプロセッサの面積の和であり，ヘテロジーニアスマルチプロセッサの面積 A_{chip} は以下のように表される．

$$A_{\text{chip}} = \sum_{j,k} A_k \, y_{j,k}.$$

これは，ヘテロジーニアスマルチプロセッサ合成問題の目的関数となる．

ノンプリエンプションにより，一つのタスクは一つのプロセッサに割り当てられる制約が生じる．それゆえに，以下の制約式が導入される．

$$\sum_j x_{i,j} = 1, 1 \leq \forall i \leq N_{\text{task}}.$$

もし，何らかのタスクがプロセッサに割り当てられれば，そのプロセッサは存在しなければならない．それゆえに，以下の制約条件が導入される．

$$x_{i,j} = 1 \rightarrow \sum_k y_{j,k} = 1$$

ヘテロジーニアスマルチプロセッサの SEU 脆弱性は，各々のタスクを実行する際の SEU 脆弱性の和である．タスクの脆弱性はタスクのプロセッサの割り当てによって決定される．それゆえに，システムの SEU 脆弱性 V_{chip} は以下のように表される．

$$V_{\text{chip}} = \sum_{i,j,k} V_{i,k} \, x_{i,j} \, y_{j,k}.$$

ここで，我々はシステム設計者が設計時に製品の SEU 脆弱性を規定すると仮定する．それゆえに，以下の制約式が導入される．

$$V_{\text{chip}} \leq V_{\text{constraint}}$$

タスク i はその到着 T_{arrival_i} 以降に実行を開始し，デッドライン時刻 T_{deadline_i} 以前に実行を終了しなければならない．よって，タスクの開始時刻を表す実数変数 s_i は以下の範囲の値をとる．

$$T_{\text{arrival}_i} \leq s_i \leq T_{\text{deadline}_i}, 1 \leq \forall i \leq N_{\text{task}}.$$

タスク i を実行するためには一定の実行時間が必要である一方，デッドライン時刻 T_{deadline_i} 以前に終了する必要がある．それゆえに，デッドライン時刻に関して，以下の制約条件が導入される．

$$s_i + \sum_{j,k} T_{\text{runtime}_{i,k}} \, x_{i,j} \, y_{j,k} \leq T_{\text{deadline}_i}, 1 \leq \forall i \leq N_{\text{task}}.$$

さて，今二つのタスク $i1$ と $i2$ がプロセッサ j に割り当てられるとし，そのプロセッサ構成はプロセッサ構成 k であるとする．これは数学的には次のように表現される．

$$x_{i1,j} = x_{i2,j} = y_{j,k} = 1.$$

二つのタスクは単一のプロセッサ上では同時には実行不可能である．二つのタスクは単一のプロセッサ上では順番に実行する必要がある．換言すれば，もし，(i) $s_{i1} < s_{i2} + t_{\text{runtime}_{i2,k}}$，かつ，$s_{i1} + t_{\text{runtime}_{i1,k}} > s_{i2}$ であれば，あるいは，もし，(ii) $s_{i2} < s_{i1} + t_{\text{runtime}_{i1,k}}$，かつ，$s_{i2} + t_{\text{runtime}_{i2,k}} > s_{i1}$ であれば，二つのタスク $i1$ 及び $i2$ は実行不可能である．逆に，二つのタスクは以下の制約条件の下で実行可能である．

$$x_{i1,j} = x_{i2,j} = y_{j,k} = 1$$

$$\rightarrow \left\{ \left(s_{i1} + T_{\text{runtime}_{i1,k}} \leq s_{i2} \right) \vee \left(s_{i2} + T_{\text{runtime}_{i2,k}} \leq s_{i1} \right) \right\},$$

$$1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}, 1 \leq \forall j \leq N_{\text{task}}, \text{ and } 1 \leq \forall k \leq N_{\text{CPU}}.$$

以上より，ヘテロジーニアスマルチプロセッサ合成問題の数学的モデルは以下のように与えられる．

Minimize the cost function $A_{\text{chip}} = \sum_{j,k} A_k y_{j,k}$
subject to
(1) $\sum_j x_{i,j} = 1, 1 \leq \forall i \leq N_{\text{task}}$.
(2) $x_{i,j} = 1 \rightarrow \sum_k y_{j,k} = 1, 1 \leq \forall j \leq N_{\text{task}}$.
(3) $\sum_{i,j,k} V_{i,k} x_{i,j} y_{j,k} \leq V_{\text{constraint}}$
(4) $s_i + \sum_{j,k} T_{\text{runtime}_{i,k}} x_{i,j} y_{j,k} \leq T_{\text{deadline}_i}, 1 \leq \forall i \leq N_{\text{task}}$.
(5) $x_{i1,j} = x_{i2,j} = y_{j,k} = 1 \rightarrow [(s_{i1} + T_{\text{runtime}_{i1,k}} \leq s_{i2}) \vee (s_{i2} + T_{\text{runtime}_{i2,k}} \leq s_{i1})], 1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}, 1 \leq \forall j \leq N_{\text{task}}, 1 \leq \forall k \leq N_{\text{CPU}}$.
Variables
- $x_{i,j}$ is a binary variable, $1 \leq \forall i \leq N_{\text{task}}, 1 \leq \forall j \leq N_{\text{task}}$.
- $y_{j,k}$ is a binary variable, $1 \leq \forall j \leq N_{\text{task}}, 1 \leq \forall k \leq N_{\text{CPU}}$.
- s_i is a real variable, $1 \leq \forall i \leq N_{\text{task}}$.

Bounds
- $T_{\text{arrival}_i} \leq s_i \leq T_{\text{deadline}_i}, 1 \leq \forall i \leq N_{\text{task}}$.

上記の制約条件 2), 3), 4), 及び, 5) は非線形式であり，MIP モデルとして解くためには線形化する必要がある．線形化を行うための標準的な手続きを適用すれば，上記の数学モデルを MIP モデルに変換することができる [25]．非線形な制約条件 2) は以下のように線形化できる．

Linearizing $x_{i,j} = 1 \rightarrow \sum_k y_{j,k} = 1$.
- $\sum_k y_{j,k} \geq x_{i,j}, 1 \leq \forall i \leq N_{\text{task}}, 1 \leq \forall j \leq N_{\text{task}}$.

制約式 3) 及び 4) の非線形項 $x_{i,j} y_{j,k}$ は以下のように線形化できる．

Linearizing $x_{i,j} y_{j,k}$.
- $z_{i,j,k} = x_{i,j} y_{j,k}, 1 \leq \forall i \leq N_{\text{task}}, 1 \leq \forall j \leq N_{\text{task}}, 1 \leq \forall k \leq N_{\text{CPU}}$.

- $z_{i,j,k} \leq x_{i,j}$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.
- $z_{i,j,k} \leq y_{j,k}$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.
- $z_{i,j,k} \geq x_{i,j} + y_{j,k} - 1$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.

また，非線形な制約式 5) は以下のように線形化できる．

Linearizing $x_{i1,j} = x_{i2,j} = y_{j,k} = 1 \rightarrow \{(s_{i1} + T_{\text{runtime}_{i1,k}} \leq s_{i2}) \vee (s_{i2} + T_{\text{runtime}_{i2,k}} \leq s_{i1})\}$.

- $\delta_{i1,i2,j,k} \leq x_{i1,j}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $\delta_{i1,i2,j,k} \leq x_{i2,j}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $\delta_{i1,i2,j,k} \leq y_{j,k}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $\delta_{i1,i2,j,k} \geq x_{i1,j} + x_{i2,j} + y_{j,k} - 2$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $s_{i1} + T_{\text{runtime}_{i1,k}} - s_{i2} - M_{i1,i2,k,1}(1 - \delta_{i1,i2,j,k}) - M_{i1,i2,k,1}(1 - \gamma_{i1,i2,j,k,1}) \leq 0$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $M_{i1,i2,k,1} = T_{\text{deadline}_{i1}} + T_{\text{runtime}_{i1,k}} - T_{\text{arrival}_{i2}}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $s_{i2} + T_{\text{runtime}_{i2,k}} - s_{i1} - M_{i1,i2,k,2}(1 - \delta_{i1,i2,j,k}) - M_{i1,i2,k,2}(1 - \gamma_{i1,i2,j,k,2}) \leq 0$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $M_{i1,i2,k,2} = T_{\text{deadline}_{i2}} + T_{\text{runtime}_{i2,k}} - T_{\text{arrival}_{i1}}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.
- $\gamma_{i1,i2,j,k,1} + \gamma_{i2,i1,j,k,2} \geq 1$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

問題 $\mathcal{P}_{\mathcal{HMS}}$ の数学モデルは以下のように表される．

Minimize the cost function $A_{\text{chip}} = \sum_{j,k} A_k y_{j,k}$

subject to

(1) $\sum_j x_{i,j} = 1$, $1 \leq \forall i \leq N_{\text{task}}$.

(2) $\sum_k y_{j,k} = x_{i,j}$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$.

(3) $\sum_{i,j,k} V_{i,k} z_{i,j,k} \leq V_{\text{constraint}}$

(4) $s_i + \sum_{j,k} T_{\text{runtime}_{i,k}} z_{i,j,k} \leq T_{\text{deadline}_i}$, $1 \leq \forall i \leq N_{\text{task}}$.

(5) $z_{i,j,k} \leq x_{i,j}$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.

(6) $z_{i,j,k} \leq y_{j,k}$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.

(7) $z_{i,j,k} \geq x_{i,j} + y_{j,k} - 1$, $1 \leq \forall i \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.

(8) $\delta_{i1,i2,j,k} \leq x_{i1,j}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(9) $\delta_{i1,i2,j,k} \leq x_{i2,j}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(10) $\delta_{i1,i2,j,k} \leq y_{j,k}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(11) $\delta_{i1,i2,j,k} \geq x_{i1,j} + x_{i2,j} + y_{j,k} - 2$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(12) $s_{i1} + T_{\text{runtime}_{i1,k}} - s_{i2} - M_{i1,i2,k}(1 - \delta_{i1,i2,j,k}) - M_{i1,i2,k}(1 - \gamma_{i1,i2,j,k,1}) \leq 0$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(13) $M_{i1,i2,k} = T_{\text{deadline}_{i1}} + T_{\text{runtime}_{i1,k}} - T_{\text{arrival}_{i2}}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(14) $s_{i2} + T_{\text{runtime}_{i2,k}} - s_{i1} - M_{i2,i1,k}(1 - \delta_{i1,i2,j,k}) - M_{i2,i1,k}(1 - \gamma_{i1,i2,j,k,2}) \leq 0$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(15) $M_{i2,i1,k} = T_{\text{deadline}_{i2}} + T_{\text{runtime}_{i2,k}} - T_{\text{arrival}_{i1}}$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

(16) $\gamma_{i1,i2,j,k,1} + \gamma_{i1,i2,j,k,2} \geq 1$, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, and $1 \leq \forall k \leq N_{\text{CPU}}$.

Variables

- $x_{i,j}$ is a binary variable, $1 \leq i \leq N_{\text{task}}$, $1 \leq j \leq N_{\text{task}}$.
- $y_{i1,i2,j}$ is a binary variable, $1 \leq i1 < i2 \leq N_{\text{task}}$, $1 \leq j \leq N_{\text{task}}$.
- s_i is a real variable, $1 \leq i \leq N_{\text{task}}$.
- $z_{i,j,k}$ is a binary variable, $1 \leq i \leq N_{\text{task}}$, $1 \leq j \leq N_{\text{task}}$, $1 \leq k \leq N_{\text{CPU}}$
- $\delta_{i1,i2,j,k}$ is a binary variable, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.
- $\gamma_{i1,i2,j,k,1}$ is a binary variable, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.
- $\gamma_{i1,i2,j,k,2}$ is a binary variable, $1 \leq \forall i1 < \forall i2 \leq N_{\text{task}}$, $1 \leq \forall j \leq N_{\text{task}}$, $1 \leq \forall k \leq N_{\text{CPU}}$.

Bounds

- $T_{\text{arrival}_i} \leq s_i \leq T_{\text{deadline}_i}$, $1 \leq i \leq N_{\text{task}}$.

以上の MIP モデルの解を求めることにより，リアルタイム制約と SEU 脆弱性制約の下でチップ面積最小なヘテロ MP を合成することができる．

4. 実　　験

4.1 準　　備

リアルタイム制約と SEU 脆弱性制約の下でヘテロジーニアスマルチプロセッサの合成を行った．ARMv4T の命令セットに準拠するプロセッサ構成を複数個用意した．全てのプロセッサ構成は仮想的なものであり，その構成は表 1 に示される．プロセッサ構成は互いにキャッシュサイズに関して異なるものである．キャッシュメモリのキャッシュヒット時の書き込みポリシーとしてライトスルー [6] を採用した．また，キャッシュライン置き換えアルゴリズムとして，LRU アルゴリズム [6] を採用した．各々の CPU コアは自身のメモリ空間を持つとし，CPU コアは互いに実行を妨げないと仮定した．キャッシュラインサイズを 32 バイトとし，セット数を 32 とした．今回の実験ではキャッシュサイズのみを変更したプロセッサ構成を用意したが，ALU 数，パイプライン段数，命令セットアーキテクチャなどの他のプロセッサパラメータを変更しても良いし，高信頼化設計を施したものを用意しても良い点に注意されたい．

表 1　実験に用いた仮想的なプロセッサ構成．

	L1 キャッシュの大きさ [kB]	仮想的なチップ面積 [a.u.]
構成 1	0	64
構成 2	1	80
構成 3	2	96
構成 4	4	128
構成 5	8	192
構成 6	16	320

実験には，組込みシステム向けベンチマークスイート MiBench から 11 種類のプログラムを使用した [5]．プログラムへの入力はその実行時間に大いに影響を与えるために，あるプログラムに対する異なる入力による実行を異なるタスクと見なした．表 2 に示すように，一つのプログラムに対して複数種類の入力を用いることにより 25 個のタスクを仮定した．本実験では，アドレストレースを生成するために，GNU の C コンパイラ及びデバッガを用いた．表 2 においては，全てのプロセッサ構成でのタスクの実行時間及び SEU 脆弱性を示す．この種の SEU 脆弱性は SEU 脆弱性見積もり手法 [1–3, 7–9, 13] を用いて容易に求めることができる．

ILOG 社の数理最適化エンジンである CPLEX 10.2 [26] を用い，3. 節に示す MIP モデルを解き，最適なヘテロジーニアスマルチプロセッサを合成した．Intel Xeon X5355 プロセッサが二つ搭載されている PC サーバを用い，最適化を行った．ほとんどの最適化プロセスにおいて，MIP ソルバはその計算を短時間で終了しなかった．それゆえに，最適化時間に 1 時間の上限を設けた．最適化を終了しなかった合成処理においては，一時解を合成結果とした．

4.2 実 験 結 果

リアルタイム制約及び SEU 脆弱性制約の下で，幾つかのヘテロジーニアスマルチプロセッサを合成した．様々な制約条件下で，ヘテロジーニアスマルチプロセッサのチップ面積を調査

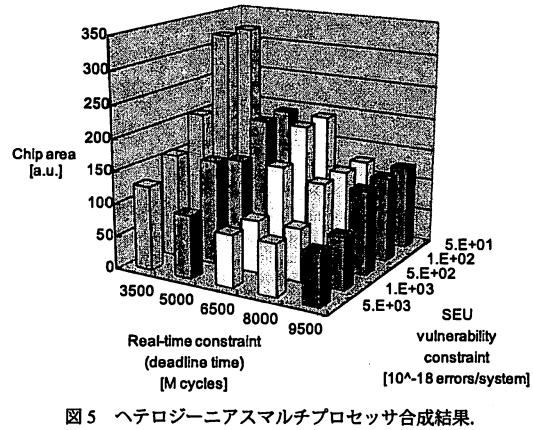

図 5　ヘテロジーニアスマルチプロセッサ合成結果．

表2 各タスクの実行時間及び SEU 脆弱性.

	タスク 1	タスク 2	タスク 3	タスク 4	タスク 5	タスク 6	タスク 7	タスク 8	タスク 9	タスク 10
Program name	bscmth_small	bitcnts_small	bf_small1	bf_small2	bf_large2	crc_small	dijkstra_small	dijkstra_large	fft_small1	fft_small2
CPU1 上での実行時間 [10^6 cycles/exec]	1980.42	293.91	328.69	1.37	2.46	188.22	442.41	2057.38	850.96	1923.92
CPU2 上での実行時間 [10^6 cycles/exec]	1011.63	53.32	185.52	1.05	1.66	43.72	187.67	832.04	412.71	935.99
CPU3 上での実行時間 [10^6 cycles/exec]	834.11	53.25	93.68	0.32	0.63	42.97	134.31	626.39	286.91	641.06
CPU4 上での実行時間 [10^6 cycles/exec]	684.62	53.15	75.03	0.26	0.51	42.97	93.31	434.72	224.98	479.29
CPU5 上での実行時間 [10^6 cycles/exec]	448.90	53.15	74.86	0.26	0.51	42.97	86.51	400.41	183.04	417.04
CPU6 上での実行時間 [10^6 cycles/exec]	405.25	53.15	74.86	0.26	0.51	42.97	83.05	382.88	182.60	417.02
CPU1 上での SEU 脆弱性 [10^{-21} errs/exec]	416.03	417.14	31.51	37.61	0.17	0.31	17.12	237.03	1141.75	356.23
CPU2 上での SEU 脆弱性 [10^{-21} errs/exec]	14025.98	96517.98	4103.81	33496.39	170.80	270.50	13217.83	27727.14	125208.68	46350.47
CPU3 上での SEU 脆弱性 [10^{-21} errs/exec]	18441.75	145977.28	9479.99	54661.44	154.06	315.47	15284.97	38577.71	181197.61	66766.15
CPU4 上での SEU 脆弱性 [10^{-21} errs/exec]	31660.22	238861.43	22248.16	70946.30	130.19	321.00	18619.48	59163.90	288057.97	113395.81
CPU5 上での SEU 脆弱性 [10^{-21} errs/exec]	50187.04	560202.80	42477.65	74006.41	135.49	336.76	19130.09	84628.95	414889.88	147621.40
CPU6 上での SEU 脆弱性 [10^{-21} errs/exec]	65564.74	653043.61	42650.39	74006.41	135.49	336.76	19300.18	172417.73	863833.06	404245.35

タスク 11	タスク 12	タスク 13	タスク 14	タスク 15	タスク 16	タスク 17	タスク 18	タスク 19	タスク 20	タスク 21	タスク 22	タスク 23	タスク 24	タスク 25
jpeg_small1	jpeg_small2	jpeg_large1	jpeg_large2	qsort_small1	sha_small	sha_large	strsrch_small	strgsrch_large	ssn_small1	ssn_small2	ssn_small3	ssn_large1	ssn_large2	ssn_large3
238.82	66.30	896.22	229.97	153.59	95.28	991.69	1.75	43.02	143.30	28.42	12.13	2043.75	849.21	226.69
86.04	32.56	319.03	111.72	75.57	20.04	208.21	1.04	23.63	30.08	11.71	5.10	390.87	379.17	105.44
58.85	18.51	270.63	59.29	46.12	17.23	177.25	0.62	14.33	20.96	7.45	2.82	282.18	245.82	58.83
52.79	14.62	198.36	51.36	45.00	17.06	173.88	0.45	10.49	20.25	5.09	2.42	279.57	148.28	43.05
51.17	14.12	192.59	50.00	44.05	16.74	173.88	0.45	10.48	20.24	5.07	2.42	279.48	147.57	43.02
50.89	14.12	191.62	49.23	43.04	16.74	173.88	0.45	10.48	20.24	5.05	2.42	279.45	147.57	43.01
1276.50	16.92	5625.82	75.59	1038.92	14.06	146.58	0.12	6.87	22.29	12.19	4.43	1617.97	3814.47	1147.60
109129.92	5330.62	1154050.94	16170.50	11847.82	3042.82	31710.01	110.65	2795.40	5280.04	1277.63	736.95	51595.47	46728.09	26758.55
159844.78	7011.33	1185073.96	20614.10	13050.32	4680.62	48761.34	161.17	5198.69	5530.73	2148.73	824.70	66569.01	93032.59	30931.43
265116.65	11887.48	115100.55	41571.20	17490.59	8848.17	92987.82	173.28	8004.63	7947.04	2483.58	1018.39	221563.88	115252.06	31531.26
303868.22	19755.82	185573.46	62095.08	22311.93	15336.85	161848.29	177.33	8764.11	16898.19	3146.46	1349.52	274845.09	137322.41	37751.81
322370.34	28336.41	248043.19	118131.10	32345.83	15358.92	162077.76	177.33	8901.50	19604.88	4656.21	1689.58	289650.63	166261.33	43999.99

した．図5は様々な制約条件下でのチップ面積を示す．図より，制約を緩和するとチップ面積が削減される傾向が分かる．

表3は，ある制約条件下でのヘテロジーニアスマルチプロセッサ合成例を示す．この合成においては，プロセッサ構成1のCPUコアを二つ，プロセッサ構成2のCPUコアを一つ持ち合わせるヘテロジーニアスマルチプロセッサが合成された．また，表に示すように、タスクはプロセッサに割り当てられた．

表3 ヘテロ MP 合成結果の一例
(T_{deadline_i} = 3.5 B cycles, $V_{\text{constraint}}$ = 500 10^{-18}errors/system).

	割り当てられたタスク
CPU 1 (構成 1)	{7, 14, 18, 21, 23, 24}
CPU 2 (構成 1)	{12, 16, 19, 20}
CPU 3 (構成 2)	{0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 15, 17, 22}

5. おわりに

本稿では，リアルタイム制約と SEU 脆弱性制約の下でのヘテロジーニアスマルチプロセッサ合成手法を提案した．ヘテロジーニアスマルチプロセッサ合成問題に対して MIP モデルを作成した．ヘテロジーニアスマルチプロセッサ合成問題を解くことにより，ヘテロジーニアスマルチプロセッサの合成を行った．実験により，リアルタイム制約，あるいは，SEU 脆弱性制約を緩和することにより，ヘテロジーニアスマルチプロセッサのチップ面積を削減できることが分かった．チップ面積と性能の間に，また，チップ面積と信頼性の間にトレードオフの関係があることが分かった．

本研究の今後の課題として，ヘテロジーニアスマルチプロセッサの合成に要する時間の削減が挙げられる．一般的に，MIP ソルバは最適解を得るまでに長い計算時間を要する．幾つかの問題に対しては，MIP ソルバは最適化を終了しなかった．MIP ソルバを使用することは，本稿のようなフィージビリティスタディには有用であるが，実用的観点からは計算時間が問題となる．より高速に合成結果を得るために，ヒューリスティックの開発が必要である．

文 献

[1] H. Asadi *et al.*, "Vulnerability analysis of L2 cache elements to single event upsets," *Proc. DATE*, pp. 1276–1281, March 2006.

[2] A. Biswas *et al.*, "Computing architectural vulnerability factors for address-based structures," *Proc. IEEE ISCA*, pp. 532–543, June 2005.

[3] V. Degalahal *et al.*, "SESEE: soft error simulation and estimation engine," *Proc. MAPLD International Conference*, 2004.

[4] P. Elakkumanan *et al.*, "Time redundancy based scan flip-flop reuse to reduce SER of combinational logic," *Proc. IEEE ISQED*, pp. 617-622, March 2006.

[5] M. R. Guthaus *et al.*, "MiBench: A Free, commercially representative embedded benchmark suite," *Proc. WWC*, 2001.

[6] J. L. Hennessy *et al.*, "Computer architecture: a quantitative approach," pp. 401–402, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[7] X. Li *et al.*, "SoftArch: An architecture level tool for modeling and analyzing soft errors," *Proc. IEEE DSN*, pp. 496–505, June 2005.

[8] S. S. Mukherjee *et al.*, "The soft error problem: an architectural perspective," *Proc. HPCA*, pp.243-247, 2005.

[9] M. Rebaudengo *et al.*, "An accurate analysis of the effects of soft errors in the instruction and data caches of a pipelined microprocessor," *Proc. DATE*, pp.10602-10607, 2003.

[10] P. Shivakumar *et al.*, "Modeling the effect of technology trends of the soft error rate of combinational logic," *Proc. DSN*, pp. 389-398, June 2002.

[11] C. W. Slayman, "Cache and memory error detection, correction and reduction techniques for terrestrial servers and workstations," *IEEE T-DMR*, 5(3):397–404, 2005.

[12] 杉原真, 石原亨, 橋本浩二, 室山真徳, "プログラムの動作を考慮したコンピュータシステムのソフトエラー数見積もり技術," 情報処理学会研究報告, Vol. 2005, No. 102, pp. 167-172, 2005 年 10 月.

[13] M. Sugihara *et al.*, "A simulation-based soft error estimation methodology for computer systems," *Proc ISQED*, pp. 196-203, March 2006.

[14] 杉原真, 石原亨, 村上和彰 "コンピュータシステムにおける信頼性と性能のトレードオフの解析と高信頼性キャッシュアーキテクチャ," 情報処理学会研究報告, Vol. 2006, No. 111, pp. 93-98, 2006 年 10 月.

[15] 杉原真, 石原亨, 村上和彰, "ソフトエラーを低減する高信頼性キャッシュメモリのためのタスクスケジューリング," 電子情報通信学会技術報告, Vol. 106, No. 386, pp. 1-6, 2006 年 11 月.

[16] M. Sugihara *et al.*, "Task scheduling for reliable cache architectures of multi-processor systems," *Proc. DATE*, pp. 1490-1495, April 2007.

[17] 杉原真, 石原亨, 村上和彰, "マルチプロセッサシステムのソフトエラー低減を目的としたタスクスケジューリング技術," 情報処理学会 DA シンポジウム, pp. 163-168, 2007 年 8 月.

[18] M. Sugihara *et al.*, "Architectural-level soft-error modeling for estimating reliability of computer systems," *IEICE Trans. Electronics*, E90-C, No. 10, pp. 1983-1991, October 2007.

[19] M. Sugihara *et al.*, "Reliable cache architectures and task scheduling for multi-processor systems," *IEICE Trans. Electronics*, Vol. E91-C, No. 4, pp. 410-417, April 2008.

[20] Y. Tosaka *et al.*, "Neutron-induced soft error simulator and its accurate predictions," *Proc. SISPAD*, pp. 253–256, 1997.

[21] Y. Tosaka *et al.*, "Simulation technologies for cosmic ray neutron-induced soft errors: models and simulation systems," *IEEE Trans. Nuclear Science*, Vol. 46, pp. 774-780, 1999.

[22] Y. Tosaka *et al.*, "Comprehensive study of soft errors in adbanded CMOS circuits with 90/130nm technology," *IEEE IEDM*, pp. 941–948, 2004.

[23] Y. Tosaka *et al*, "Comprehensive soft error simulator NISES II," *Proc. SISPAD*, pp. 219–226, 2004.

[24] N. J. Wang *et al.*, "Characterizing the effects of transient faults on a high-performance processor pipeline," *Proc. DSN*, pp.61-70, 2004.

[25] H. P. Williams, *Model Building in Mathematical Programming*, John Wiley & Sons, 1999.

[26] ILOG Inc., CPLEX 10.2 User's Manual, March 2007.

[27] International Technology Roadmap for Semiconductors, International Technology Roadmap for Semiconductors 2003 Edition, *http://public.itrs.net/*, 2003.

[28] International Technology Roadmap for Semiconductors, International Technology Roadmap for Semiconductors 2005 Edition, *http://public.itrs.net/*, 2005.