Vol.2013-SLDM-160 No.23
Vol.2013-EMB-28 No.23
2013/3/14

CSP 理論にもとづいた制御モデルのマルチコア実装向けタスク割当て

大川 禎[1]　枝廣 正人[1]　久村 孝寛[2]

概要：近年，マルチコア・メニーコアが組込みシステムにおいても主流となりつつある．また，制御処理を記述する上で有効なソフトウェアモデルとして，CSP（Communicating Sequential Processes）があげられる．本論文では，汎用マイクロコントローラベースのマルチコアシステムをターゲットとし，CSP により記述されたモーター制御モデルの実装を行い，実行時間が最小となるタスク割当てパターンを非線形計画問題によって発見した．また，従来手法から得られたタスク割り当てパターンと性能比較を行った結果，従来手法に比べ 15%性能を改善することができた．

1. はじめに

近年，CPU 設計の大規模複雑化，消費電力の増大などの問題により，単一 CPU の性能向上は限界に到達しようとしている．そのため，現在は，CPU 内に複数のプロセッサコアを搭載したメニーコアが組込みシステムにおいても主流となりつつある．しかし，プロセッサのメニーコア化によるハードウェアの性能向上だけではシステムの高速化は期待できず，メニーコアリソースを効率的に活用するために実装するソフトウェアが並列化を意識して設計されていなければならない．並列化が有効なソフトウェアモデルとして，CSP（Communicating Sequential Processes）[1] があげられる．CSP は，システム全体をイベント駆動によって独立して実行されるプロセスの集合として並行処理システムを表現する．内部の処理と非同期に入力を受け付けるようなシステムや，リアルタイム性が要求されるシステムを記述する上ではモデル記述が容易である．本論文では，汎用マイクロコントローラベースのマルチコアシステムをターゲットとし，CSP により記述されたモーター制御モデルの実装を行う．

対象とするアーキテクチャでは，異なるプロセッサ間の通信においては通信コストが生じ，実行性能に影響を与える．そして，実装するモデルの実行時間の内，通信コストが占める量はタスク割当てパターンにより変化する．また，タスクの依存関係や実行順序を考慮したタスク割当てを行うことで，タスクの実行が待たされることによるプロセッサの空き時間を減らすことができる．したがって，実装するモデルが高い実行性能を得るためにタスク割当てが重要となる．

そこで，ターゲットへのタスク割り当てを非線形計画問題として実行時間を定式化することで，非線形計画問題ソルバにより実行時間が最小となるタスク割当てパターンを発見する．また，従来手法としてグラフ分割アルゴリズムである hMETIS[2] から得られるタスク割当てパターンと，本論文で得られる割当てパターンそれぞれにおいて実行時間を計測し，性能を比較することで評価を行う．評価実験はルネサス エレクトロニクス社の V850 コアシミュレータ [3] を用いて行う．

評価の結果，モータ制御において従来手法によるタスク割当てパターンに対し最大 15%の性能改善を達成し，また，4 コアでの実行時間がシングルコアに比べ 22%短縮されたことを示す．

2. CSP

2.1 概要

CSP モデルとは，並行システムにおける相互作用のパターンを記述する仕様記述言語 CSP によって記述されるモデルである．複数のプロセスと，プロセス間の通信路であるチャネルによってシステムが定義される [1]．各プロセスは独立して逐次実行され，他プロセスとメッセージ通信を行う．通信を必要とするプロセス間にはチャネルが定義され，プロセス間で通信されるデータはチャネルを介して送受信される．

CSP システム全体は複数のプロセスに分割することがで

[1] 名古屋大学 情報科学研究科
Graduate School of Information Science, Nagoya University
[2] 日本電気株式会社 グリーンプラットフォーム研究所
Green Platform Research Laboratories, NEC Corporation

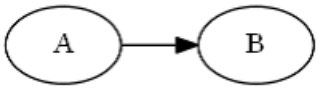

図 1 CSP モデル例

き，各プロセスは独立して逐次実行される．プロセス間で依存するデータは，チャネルによってプロセス間で受け渡される．例えば，あるプロセス A が変数 n を操作し，別のプロセス B が操作された n を必要としているとき，プロセス B はプロセス A に依存し，プロセス A はチャネルを介してプロセス B へ変数 n を送信する．

CSP モデルを視覚化すると，図 1 のようなデータフローダイアグラム (DFD) で表せる．

楕円をプロセス，楕円をつなぐ矢印がチャネルを表す．図 1 では，2 つのプロセスと 1 つのチャネルが定義されている．プロセス B が必要としているデータはプロセス A がチャネルを介してプロセス B へ受け渡している．図 1 のチャネルは，プロセス A の出力チャネルであり，プロセス B の入力チャネルである．

2.2 プロセス間の通信

プロセス間の通信はチャネルによって行われ，チャネルによる通信は次の 3 つの特徴を持つ．

(1) チャネル通信はバッファを持たない
(2) チャネルの接続形態は一対一
(3) データの流れは単方向

1 の性質から，プロセス間のデータの送受信は同期して行われる．例えば，送信側のプロセスがチャネルへデータを送信できる（Ready）状態になった際に受信側のプロセスが受信できる（Ready）状態でないとき，通信するデータを蓄積させて処理を続行することができないため，受信側のプロセスが Ready 状態になるまで送信側のプロセスは待たされることになる．逆に，受信側のプロセスが Ready 状態にあるとき，受信されるべきデータは蓄積されてない（バッファが存在しない）ため，送信側のプロセスが Ready 状態になるまで受信側のプロセスは待たされることになる．したがって，チャネル通信における送信と受信はタイミングが同期される．

2.3 想定するマルチコアアーキテクチャ

図 2 に本論文で想定するマルチコアアーキテクチャを示す．各コアにキャッシュおよびローカルメモリを持ち，コア間は共有メモリによる通信を行う．共有メモリ間通信においては，共有メモリで共有しているデータの準備が整ったことだけを通信で伝えているため，通信時間は一定の値となる．また，制御処理をターゲットとするため，ノンコヒーレント・キャッシュを想定する．

このような汎用マルチコアシステムへ CSP モデルを実

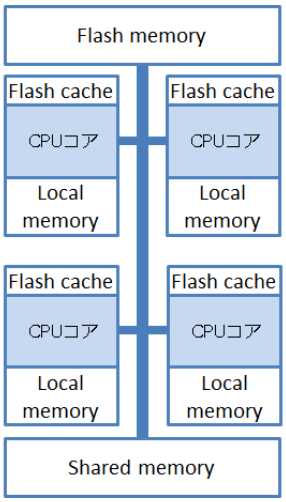

図 2 想定するマルチコアアーキテクチャ

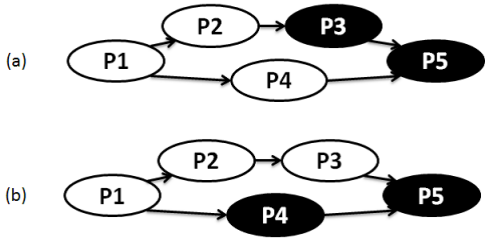

図 3 タスク割り当て例

装するとき，各タスクを実行するコアがそれぞれ割り当てられ，各コアでは複数のタスクをタスクの優先度順に実行する．タスクの優先度はトポロジカルオーダーにより決定され，入力側に近いタスクから順に高い優先度を持つ．プロセス間は共有メモリによる通信が行われ，異なるプロセッサ間の通信においては通信コストが発生する．これにより，タスク割当てパターンによりよって実装するモデルの実行時間に差が発生する．

次に，タスク割り当てパターンにより実行時間が変化する例を示す．図 3 はある CSP モデルを表し，(a) と (b) で割り当てパターンが異なることを示している．このモデルは 5 つタスクをもち，各タスクは P1,P2,P3,P4,P5 の順に優先度が高い．同色で示されたタスクは同じコアに割り当てられているものとする．(a) と (b) では P3 と P4 の割り当てられているコアが異なっている．図 3 の (a),(b) それぞれを実行したときの実行順の例を図 4 に示す．P1〜P5 を長方形で示し，長方形の横の長さはそのタスクの実行時間を示す．ここで，(a),(b) はそれぞれ同じモデルを実行しているが，(b) のほうが早くすべてのタスクが実行されることがわかる．

このように，CSP 理論に基づき記述されたモデルの実装においては，タスク割り当てにより実行時間が変化することがわかる．したがって，実装するモデルから高い性能を引き出すためには実行時間が短くなるようなタスク割当て手法を検討する必要がある．

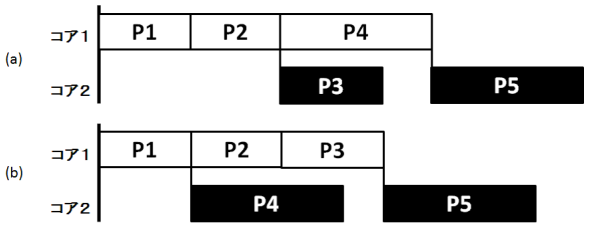

図 4 タスク実行例

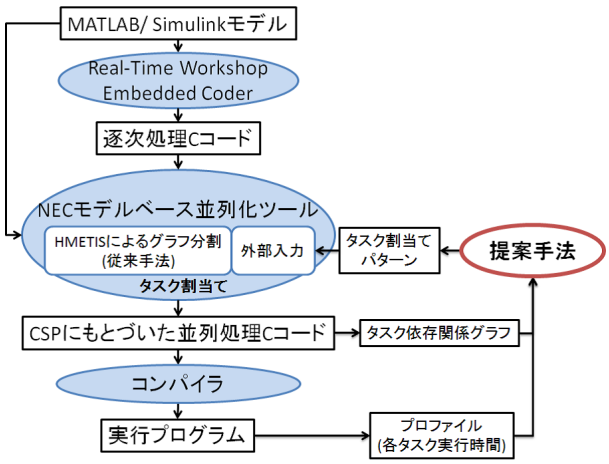

図 5 制御モデルからコード生成までのフロー

3. 制御モデルからのコード生成フロー

本節では，制御モデルからコード生成までのフローを述べる．本論文では制御モデルとして MATLAB Simulink モデル [4] を仮定する．図 5 にフローの概要を示す．

MATLAB Simulink により記述された制御モデルは MATLAB の機能である Real-Time Workshop Embedded Coder により，逐次 C コードへ変換される．なお，この C コードはマルチコアで実行可能な並列処理として記述されていない．そのため，この逐次コードはマルチコアにおいては高い性能を発揮できない．そこで，モデルベース並列化ツール [5] を用いて，Simulink モデルと逐次処理 C コードから，マルチコア向けに記述された並列処理 C コードに並列化する．これは CSP 理論に基づき記述されたモデルとなっている．そして，並列処理 C コードをコンパイルすることでマルチコア環境でタスクを並列実行できるターゲットコードが生成される．その際，並列化ツールにタスク割当てパターンを入力することで，割当てを反映させたコードを出力することができる．本論文では，各タスクの逐次処理実行時間をプロファイルとし，タスク間依存関係グラフとプロファイル結果から，提案手法によって実行時間が最小となるタスク割当てパターンを発見する．

なお，モデルベース並列化ツールが Simulink モデルから並列 C コードを生成する手順は，以下のステップで構成される．

(1) C コード解析 Simulink から出力された逐次処理 C コードを基本ブロックに分割．

(2) モデル解析 Simulink モデルから，ブロックの接続関係，ブロックの階層構造を読取る．

(3) 中間モデル構築逐次処理 C コードとモデルの解析結果にもとづいて，並列 C コード生成用の中間モデルを作成．

(4) 並列コード生成ターゲット OS に合わせて，中間モデルから並列 C コード生成．

また，本論文で想定するアーキテクチャを対象としたコード生成時には，各タスク内のローカル変数はローカルメモリに配置され，複数のタスクが共有するデータは共有メモリに配置される．

4. 非線形計画問題による定式化

モデルの実行時間を非線形計画問題として定式化する．実行時間は，先頭のタスクの開始時刻に対し，最後に実行されるタスクの完了時刻である．モデルのタスク数を N，割り当てコア数を M としたとき，モデルの実行時間は，各タスクの実行時間 $C_i(i = 1, 2, \cdots, N)$，および異なるプロセッサ間の通信時間 T の和で表せる．また，タスク間依存関係は隣接行列 $A_{ij}(i = 1, 2, \cdots, N, j = 1, 2, \cdots, N)$ で表す．タスク i がタスク j に依存しているとき，$A_{ij} = 1$ となる．なお，本論文では便宜上タスク番号とタスクの優先度を同一のものとみなす．タスク番号が小さいほど優先度が高いものとする．

この問題においては，各タスクの割り当てコアが変数 $X_i(i = 1, 2, \cdots, N)$ となる．割当てコアは有限であるから，変数 X_i において (1) 式が制約条件となる．

$$X_i = 1, 2, \cdots, M \tag{1}$$

ここで，先頭のタスクの開始時刻を 0 としたとき，タスク i の完了時間 E_i は i の直前に実行されるタスクの完了時間と，i の実行時間の和である．

自身の直前に実行される可能性があるタスクは，i に依存しているタスク，または，i より優先度が高いかつ i と同じプロセッサで動作するタスクである．i に依存しているタスクを j とすると，$A_{ji} = 1$ が成り立つ．また，i より優先度が高いかつ i と同じプロセッサで動作するタスクを k とすると，$k < i$ かつ $X_i = X_j$ が成り立つ．

i の直前に実行完了されるタスクは上記に示した自身の直前に実行される可能性があるタスクのなかで，完了時刻が最大となるものである．よって，E_j をタスク i の直前に実行されるタスク j の完了時間とすると，(2) 式が成り立つ．

$$E_i = \max(E_j + t | A_{ji} = 1 \ or \ j < i \ and \ X_i = X_j) + C_i \tag{2}$$

ここで t は次の値を表す．

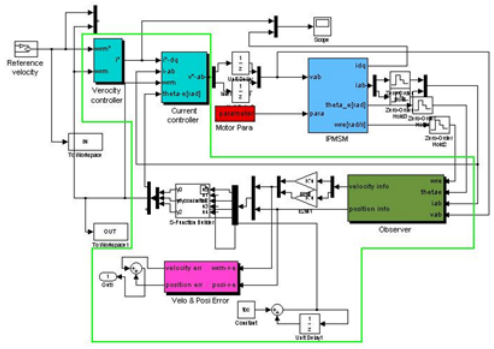

図 6 モーター制御を行う Simulink モデル

$$t = \begin{cases} T & (a_{ji} = 1 \; and \; X_i \neq X_j) \\ 0 & (otherwise) \end{cases} \tag{3}$$

また，各タスクの完了時間 E_i はタスク優先度順に計算することが可能である．

各タスクの完了時間のうち，最大値をとるタスクが最後に完了するタスクであるから，目的関数は (4) 式で表せる．また，目的関数を最小化する変数が実行時間最小となるタスク割当てパターンである．

$$\max(E_i, i = 1, 2, \cdots, N) \tag{4}$$

したがって，以下のようにモデルの実行時間を非線形計画問題として定式化することが可能である．

$$\text{minimize}$$

$$\max(E_i, i = 1, 2, \cdots, N) \tag{5a}$$

$$\text{subject to}$$

$$X_i = 1, 2, \cdots, M \tag{5b}$$

4.1 ソルバ

先に述べた非線形計画問題の解を導出するために，Excel2010 ソルバアドインを用いた．これは線形計画問題，非線形計画問題を解くことができるソルバで，非線形計画問題においては局所最適解を得る [6]．

5. 評価実験

5.1 評価環境

本実験における評価環境は以下のとおりである．

CPU Renesas V850 コアシミュレータ

RTOS eSOL MCOS ver.0.4

また，前節で得られたタスク割り当てパターンの有効性を検証するために，評価用ベンチマークとして図 6 に示すモーター制御モデルを CSP 記述したものを実装した．図 7 に DFD を示す．

これはフィードバック制御を行うモデルである．フィードバック制御モデルは，外界からの信号を入力値として制御値を出力し，制御した結果を入力側へ戻す処理を行うものである．外界からの入力値に対し出力制御値を決定するコントローラと，コントローラから得られた信号を入力とし，フィードバック値を出力としてコントローラへ返すプラントで構成される．フィードバック制御モデルへは複数回の入力が行われ，1 入力に対しコントローラ，プラントそれぞれで内部処理が行われる．プラントは，1 入力に対し制御する機構の内部処理を行い，コントローラへ返すフィードバック値を出力する．コントローラは，モデルへの入力値と，前回の入力に対するフィードバック値に対し，出力をプラントへ渡す．

図 7 CSP に基づき記述したモーター制御モデル

CSP に基づき記述されたモデルにおいては，先頭のタスクがモデルへの入力を受け取り，すべてのタスクが実行されるとフィードバック値が出力される．先頭のタスクが開始され，すべてのタスクが実行完了するまでを 1 ステップとする．

5.2 実験方法

本実験では，上記のモデルから 2 つのベンチマークを生成した．

(1) コントローラ部のみ

(2) コントローラ・プラント

それぞれのベンチマークを 1000 ステップ実行し，各ステップにおいて実行時間を計測した平均値を 1 ステップの実行時間とした．

提案手法による解を導出するにあたり，本実験では通信時間を $T = 100$ と見積もった．これは，シミュレータ上で二つのプロセッサ間でデータ準備完了の通信をする際のサイクル数を計測したときの平均値である．

また，本論文における提案手法に加え，従来手法として，タスク配置問題をグラフ構造の分割問題とみなし，グラフ分割アルゴリズムである hMETIS を用いたタスク割当てを行った．hMETIS は大規模なグラフにおいても処理時間が速いという特徴があり，LSI の回路分割などに応用されている．

さらに，従来手法，提案手法それぞれにおいて，M = 2,3,4 における最適な割当てを計算し，M=1 における実行時間を 1 としたときの実行時間比を計算した．

さらに，提案手法において最小化された目的関数の値を実行時間理論値として求め，同様に M=1 における実行時間理論値を 1 としたときの実行時間比を計算し，実際の実行時間比と比較した．

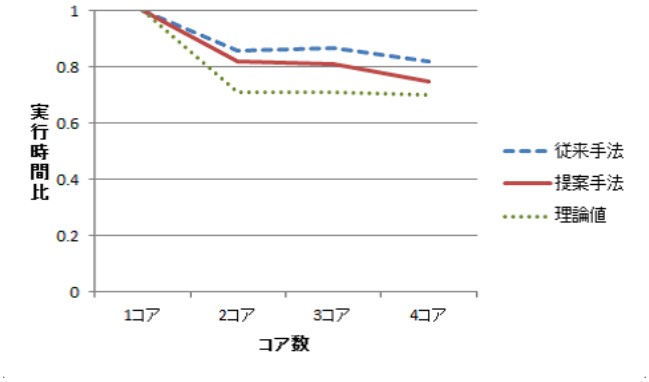

図 8 コントローラにおける実行時間比

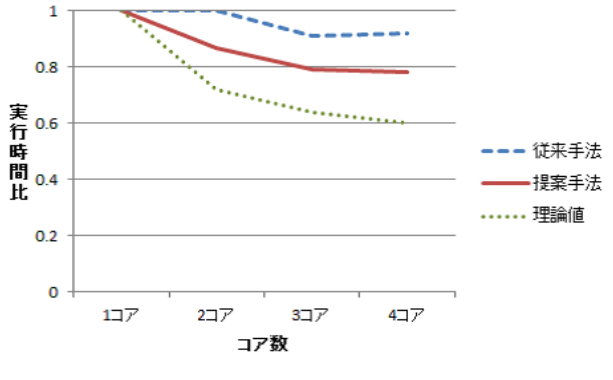

図 9 プラント，コントローラにおける実行時間比

6. 実験結果

表 1 は，コントローラ部のみにおける従来手法，提案手法による実行時間比，および提案手法から得られた実行時間理論比を表している．また，表 2 のグラフを図 8 に示す．

表 1 コントローラにおける 1 ステップの実行時間比

	1 コア	2 コア	3 コア	4 コア
従来手法	1	0.86	0.87	0.82
提案手法	1	0.82	0.81	0.75
理論値	1	0.71	0.71	0.70

従来手法においては 4 コアでの実行時間比が 0.82 となっていることから，シングルコアに対し 18%性能が向上している．また，提案手法においては 4 コアで 25%性能向上が得られた．

従来手法と提案手法を比較すると，全体的に提案手法では従来手法に比べ良い性能が得られていることがわかる．また，コア数の増加に応じて性能差は開き，4 コアで最大 9%となった．また，理論値をみると 2 コア以上で実行時間比はほぼ同じ値となっていることから性能向上は 2 コアにおいてほぼ限界に達している．

表 2 プラント，コントローラにおける 1 ステップの実行時間比

	1 コア	2 コア	3 コア	4 コア
従来手法	1	1.00	0.91	0.92
提案手法	1	0.87	0.79	0.78
理論値	1	0.72	0.64	0.60

従来手法においては 4 コアでの実行時間比が 0.92 であり，シングルコアに対し 8%性能が向上している．また，提案手法においては 4 コアで 22%性能向上が得られた．

プラント・コントローラにおいても提案手法のほうが高い性能を示し，提案手法は従来手法に比べ最大 15%性能が改善している．

6.1 考察

従来手法に比べ提案手法で性能向上が見られた理由として，領域分割によるタスク割り当てを行う従来手法では，領域分割ではタスクの実行順序が考慮されていないこと，領域数とプロセッサ数が一対一対応していて細かな割当て制御をやりにくいことが考えられる．

2〜4 コアで性能向上が限界を迎える理由として，クリティカルパス上のタスクの負荷が大きいために，全体で並列実行される割合が低い事が考えられる．コントローラモデルにおいては全タスクの総実行時間のうちクリティカルパス上のタスクの総実行時間が 67%となっている．また，プラント・コントローラモデルにおいても全体の 59%であり，コントローラ部のみに比べて並列度は高いが，理論上 4 コア以上でほぼ性能限界に到達している．

また，並列度を下げる要因として，タスクの計算時間の粒度が粗い事が考えられる．計算量の多いタスクがボトルネックとなっている可能性がある．

また，理論比と実際の性能比の違いとして，いくつか原因が考えられる．本実験でプロファイルとして計測したタスクの実行時間はモデル全体を逐次実行したもので，並列処理においてはメモリアクセスやバストラヒックの影響を受けるため，実際のタスク実行時間がプロファイル結果と異なっていることが考えられる．そして，タスクの実行時間は動的に変化し一定値をとらないことから，プロファイルによって得られた理論的な最適解が実際の最適解とはならない可能性が考えられる．しかし，本論文では理論値と実測値の差を生む明確な要因分析はできていないため，今後の課題とする．

6.2 補足実験

さらに，補足実験として提案手法で得られた解の妥当性を検証するために，2 コア割当てにおいて全数探索による実行時間最小となる割り当てパターンを発見した．提案手法では非線形計画問題ソルバによって実行時間が最小となるタスク割り当てパターンを発見したが，本実験ではすべてのタスク割当てパターンにおいて 1 ステップの実行時

間を計算し，最小値となるタスク割り当てパターンを発見した．

その結果，全探索により非線形計画ソルバから得られた解とは異なる解が得られた．しかし，全探索解による実行時間を計測したところ，非線形計画ソルバによる解よりも実行時間が長くなる結果が得られた．理論値と実際の実行時間が合わない理由として，上記と同様に通信コストの設定が正しくないことにより不正確な理論値を出していると考えられる．ただ，全探索および非線形計画ソルバの解はそれぞれ実行時間に差は発生しなかったため，非線形計画ソルバによる解は妥当なものであると考えられる．

7. まとめ

従来手法に比べ，タスク割当てパターンの改善により性能向上が期待できる事がわかった．また，今回定式化した非線形計画問題から得られる割り当てパターンは実際の実行時間を反映していない．より高い精度で実行時間を見積もるためには，メモリアクセス，バストラヒックの影響を考慮する必要があると考えられる．

謝辞 本論文を進めるにあたり，トヨタ自動車株式会社，ルネサス エレクトロニクス株式会社，イーソル株式会社の多大なる協力をいただいた．ここに深く感謝の意を表する．

参考文献

[1] Hoare, C. A. R. *Communicating Sequential Processes*. the United Kingdom, Prentice Hall, 2011.
[2] George Karypis and Vipin Kumar, hMETIS A Hypergraph Partitioning Package Version 1.5.3. University of Minnesota, Department of Computer Science & Engineering Army HPC Research Center, 1998.
[3] ルネサス エレクトロニクス, リアルタイム制御システムに適した V850 CPU 向け仮想化技術を開発, http://japan.renesas.com/press/news/2010/news20100929.jsp, 2013/02/10.
[4] Simulink - シミュレーションおよびモデルベース デザイン (MBD) - MathWorks, http://www.mathworks.co.jp/products/simulink/, 2013/02/07.
[5] Takahiro KUMURA Yuichi NAKAMURA, Nagisa ISHIURA, Yoshinori TAKEUCHI, Masaharu IMAI, Model Based Parallelization from the Simulink Models and Their Sequential C Code. SASIMI 2012 Proceedings.
[6] Frontline Systems, Inc. http://www.solver.com/.