組込みメニーコアに向けた（ＯＳと）制御アプリ並列化

２０１３年8月２３日

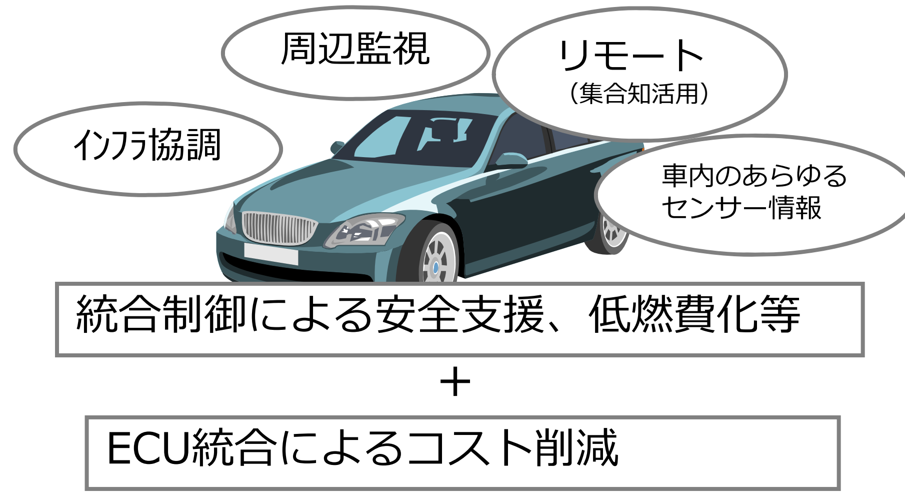
名古屋大学大学院情報科学研究科
東京大学大学院情報理工学系研究科
早稲田大学グリーン・コンピューティング・システム研究開発センター
枝廣　正人

アウトライン

- 背景
  - マルチ・メニーコアの動向（JEITA調査より）
- MATLAB/Simulinkモデルからの並列化
- 車載制御システムを例とした並列化と課題
- まとめと今後の課題、取り組み

車載システムのマルチ・メニーコア実現の必要性

<u>自動車業界のニーズ</u>
演算量の増加とリアルタイム処理の両立
－高機能化、複合センサー処理
　ECU統合によるコスト削減ニーズ　など

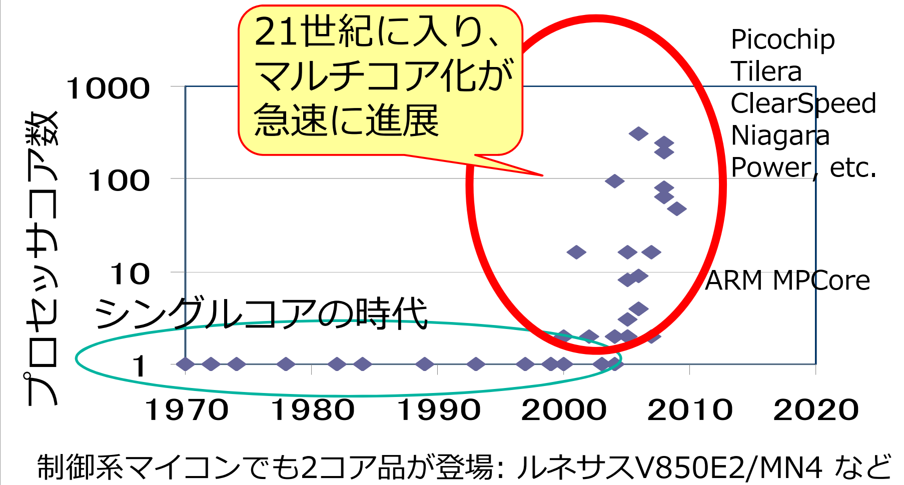


統合制御による安全支援、低燃費化等

＋

ECU統合によるコスト削減

<u>技術シーズ</u>
CPUはマルチコアが主流へ
－シングルコアでの性能向上の限界
　（熱、消費電力の課題など）



制御系マイコンでも2コア品が登場：ルネサスV850E2/MN4 など

ソフトウェアの並列化が必要不可欠
⇒モデルベース設計からの並列化

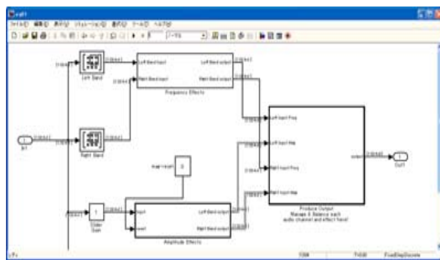
アウトライン

- 背景
- <u>MATLAB/Simulinkモデルからの並列化</u>
- 車載制御システムを例とした並列化と課題
- まとめと今後の課題、取り組み

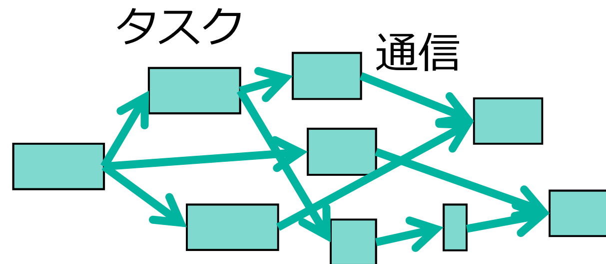
モデルベース並列化: Simulinkモデルから並列コードを生成 (NECとの共同研究)

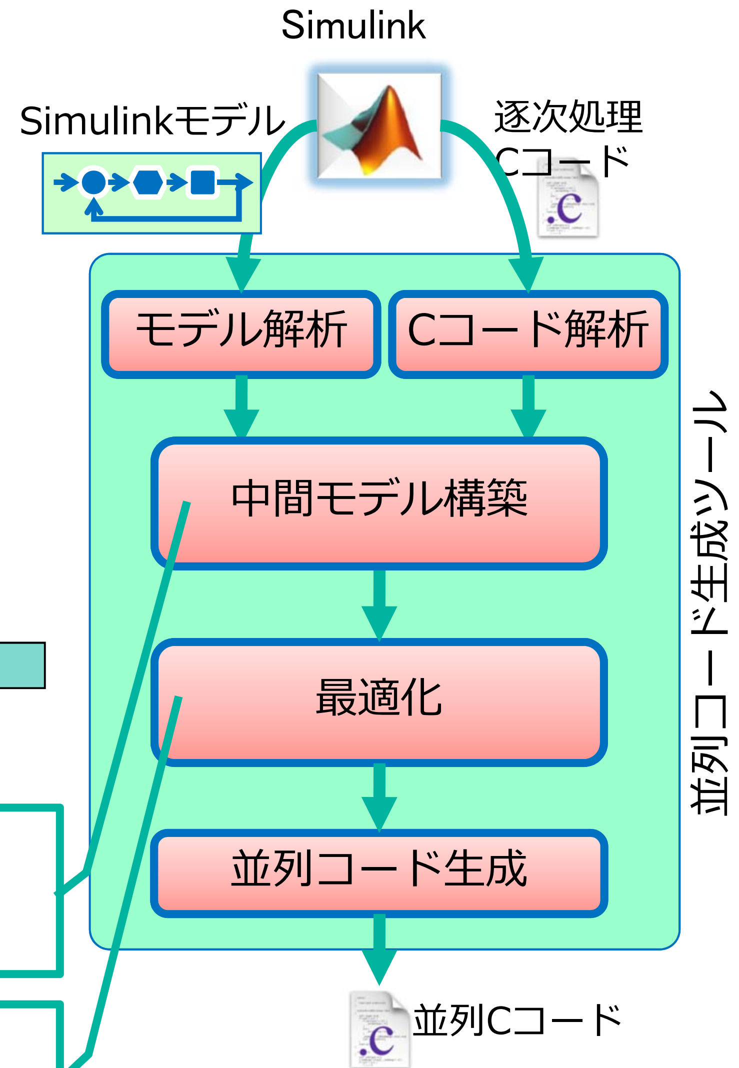
- ブロック図で表記されたモデルから並列Cコードを生成
- 並列APIを明示的に使うことなく、 並列プログラムを開発可能に

並列化手法：
- モデルのブロックをタスクにマッピング。
- タスクの実行完了をメッセージで通知。



Simulinkモデル





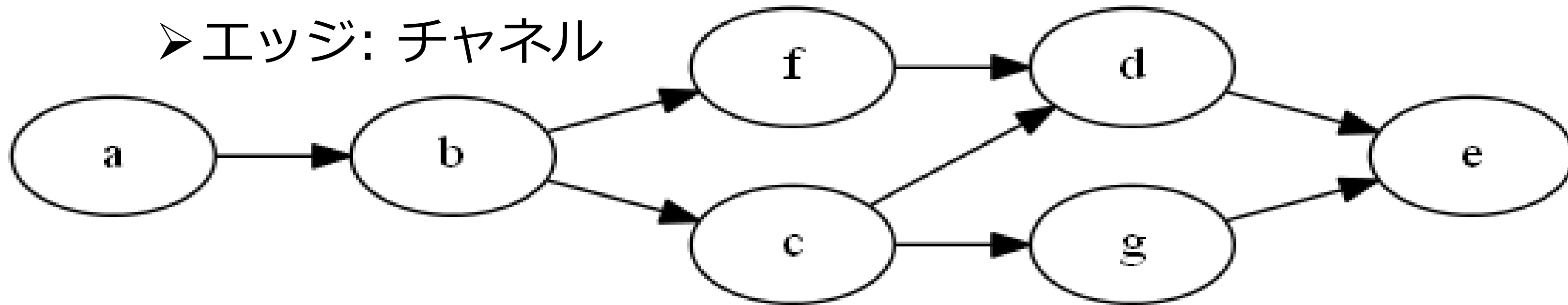
モデルに表現された構造的並列性を最大限抽出
例）フラット化、ループ構造分解

通信オーバーヘッド低減のため、タスク粒度調整、タスク完了待合せ、などの最適化

*1: The MathWorks社のモデルベース開発ツール。
*2: RealTime-Workshopが生成したCコード

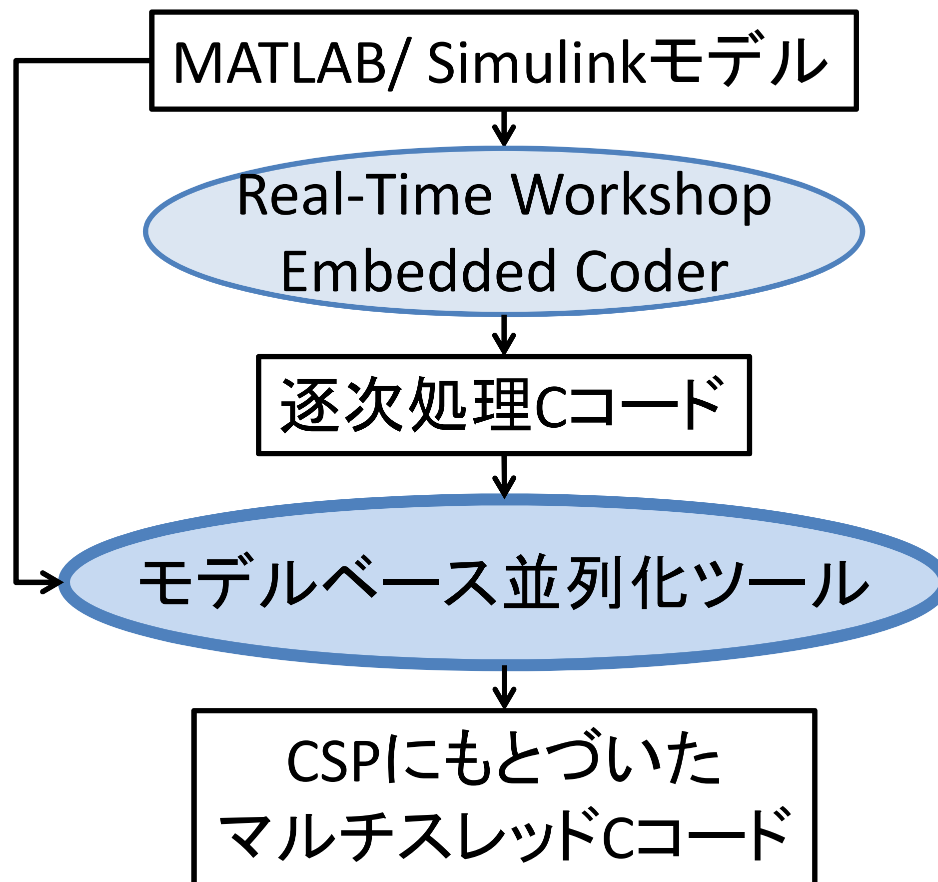
CSP並列実行モデル

- CSP(Communicating Sequential Processes)
  - 並行実行するプロセス(タスク)の相互作用を記述
    - ノード: プロセス
    - エッジ: チャネル



- 制御処理と相性が良い
  - センサ入力値に対して制御値を出力する処理がCSPの動作に当てはまっている

モデルベース並列化ツールを用いたフロー



参考文献:久村孝寛. Simulinkモデルにもとづいた並列Cコード生成.
電子情報通信学会技術研究報告, Vol.110, No. 473, pp. 303–308, 2011.

モデルベース並列化の特徴

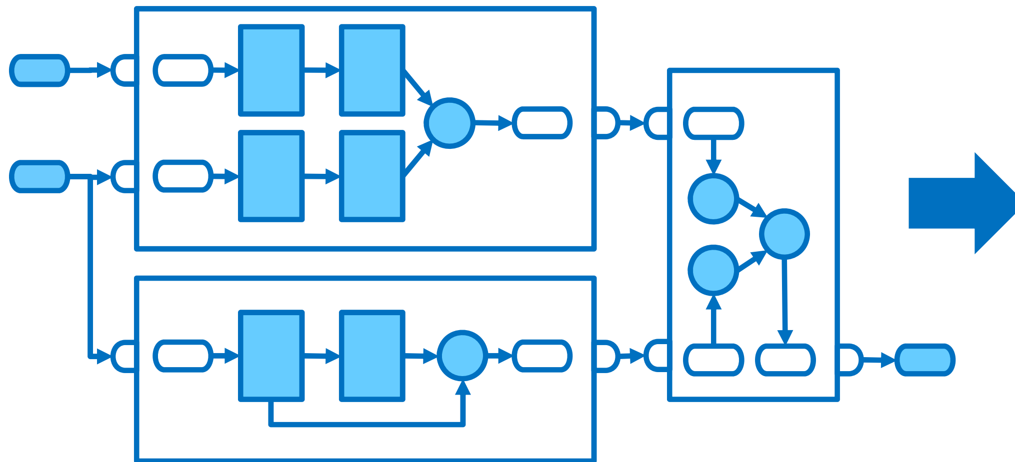
- **データフローにもとづいた並列処理**
  - ブロック図で表現されたモデルの、ブロックをタスク、ブロック間の線をデータ依存関係、と見なして、タスク間のデータ依存関係を満たすような並列処理を実施。
  - **共有メモリによるデータ共有**
    - 通信によるデータ転送増加を防ぐために、共有メモリを使ってタスクはデータを共有。
  - **1-1通信だけを使用**
    - 仕様が複雑になりがちなN-1通信を使わず、デッドロックが発生しないように通信の順番を決めた上で、1-1通信だけを使う。
    - ターゲット・デバイスがN-1通信を提供していれば、N-1通信を使うことも可能。
  - **並列化方法を選択可能**
    - ターゲットプロセッサの特性に合わせて並列実行方法を変更可能。

- **モデルから並列性を抽出するための処理**
  - **フラット化**
    - モデルの階層構造を無くして、ブロックの接続関係を管理し易くする。
  - **ループ構造除去**
    - 遅延要素ブロックに着目し、並列処理に適すように、並列化を困難にするループ構造をモデルから除去する。
  - **グループ化**
    - タスクやチャネルの増加によるオーバーヘッド増加を防ぐために、計算量が小さなタスクをグループ化し、タスクやチャネルの数を減らす。

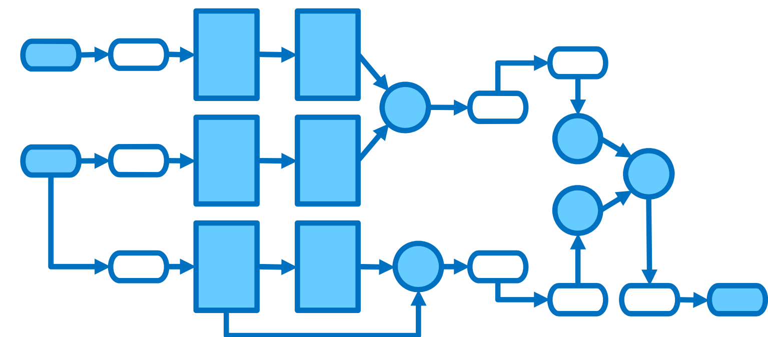
フラット化

- Simulinkモデルは階層的な構造をもつ。
- 並列化しやすいよう、階層構造を分解。

階層構造をもつモデル



フラット化されたモデル



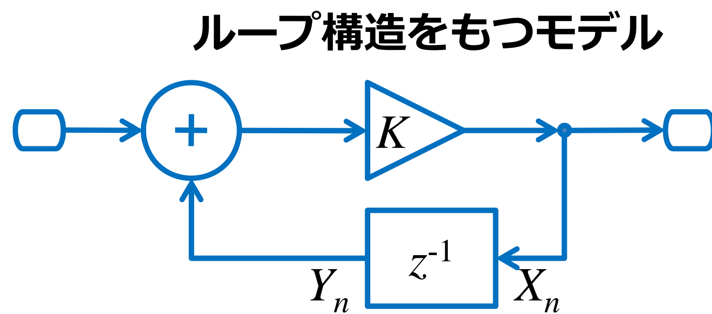
対応するCコードが存在するブロック

定数や入出力端子等に対応するCコードは、
定数や変数として埋め込まれ、
逐次Cコードの中に陽に現れないことがある。

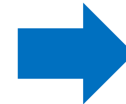
全てのブロックの依存関係をひとつのグラフ構造で表現し、ここから並列性を抽出

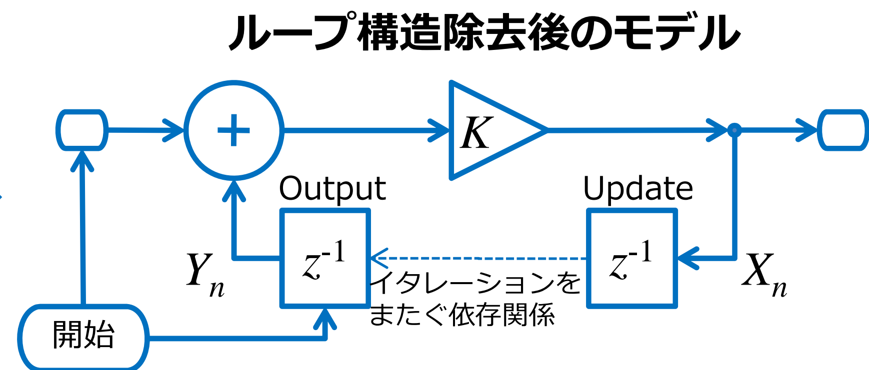
ループ構造除去

- ループ構造は並列化を阻害するため、遅延要素ブロックを出力と更新に分割し、ループ構造を除去。



遅延要素の出力: $Y_n = f(S_n)$
遅延要素の状態更新: $S_{n+1} = g(X_n, S_n)$





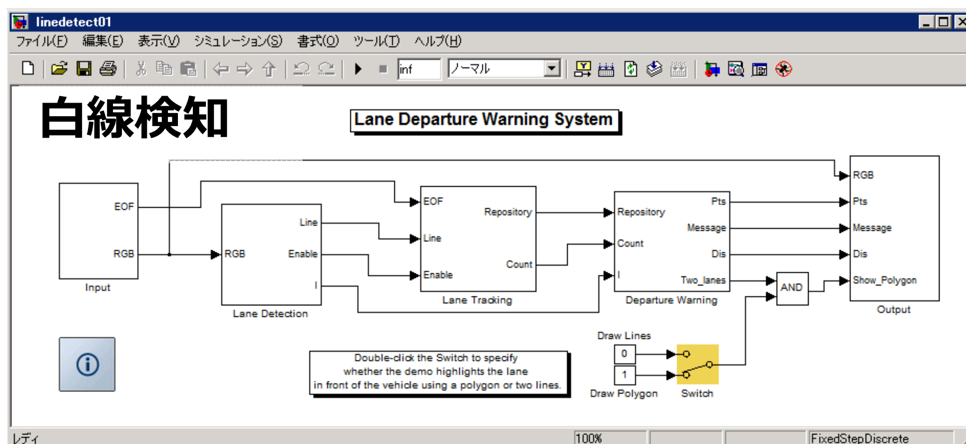
出力Y_nは入力X_nに依存しないので、
Xnの到着を待たずにY_nをすぐに計算可能。
Y_nを計算した後で、状態S_{n+1}を計算すればよい。

遅延要素をOutput(出力)とUpdate(更新)に分割。
開始ノードからOutputへのエッジを追加。
Updateの結果(S_{n+1})を次のイタレーションにおいて
Outputが使用するので、イタレーションをまたぐ
依存関係を表すエッジをUpdateからOutputへ追加。

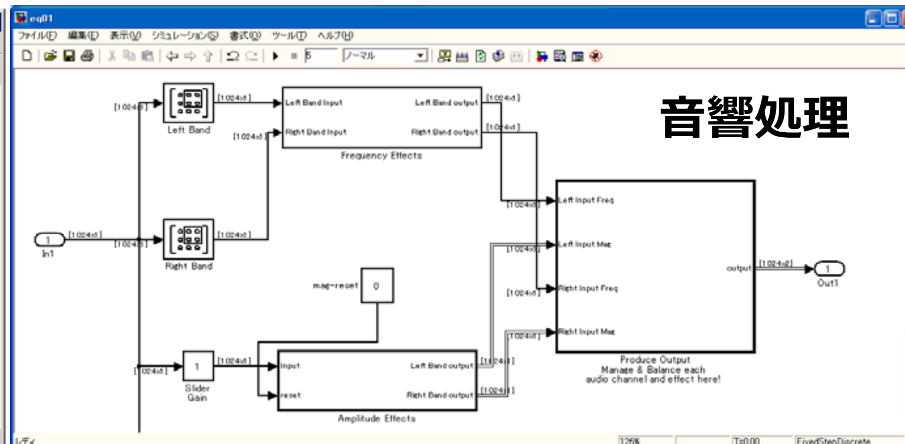
イタレーションをまたぐエッジ(破線)を無視すると、
ループ構造はなくなっている。

モデルベース並列化による並列化実験

一般に公開されているSimulinkモデルから並列コードを生成し、効果を確認



道路画像から白線を検出し、追尾し、白線を逸脱したら警告メッセージを出力する。
MATLAB の Video and Image processing toolbox に付属するサンプルモデル: vipldw_all.mdl



様々なエフェクトをかけて、オーディオ信号(ステレオ)の振幅特性や周波数特性を変化させる。
http://www.mathworks.com/matlabcentral/fileexchange/18317-professional-simulink-audio-equalizer

モデル	ブロック数	タスク数	実行時間(並列化前を100%とする)		
			Windows Xeon 4コア@1.8GHz		eSOL eT-Kernel SMP NaviEngine (4コア@400MHz)
			Intelコンパイラ 自動並列化	モデルベース並列化 (パイプライン的並列処理、未グループ化)	
音響処理	252	57	85.8%	38.3%	26.3% ・フィードバックループがない。・各タスクの計算量がそれなり。
白線検知	302	64	94.9%	44.3%	39.0% ・タスクの計算量ばらつきが大きい。

T. Kumura, et al., "Model Based Parallelization from the Simulink Models and Their Sequential C Code," Proceedings of SASIMI 2012, R2-8, pp. 186-191, March 2012.

アウトライン

- 背景
- MATLAB/Simulinkモデルからの並列化
- 車載制御システムを例とした並列化と課題
  - MATLAB/SimulinkモデルからCSPモデルへ
  - V850マルチコアアーキテクチャでの実現
  - XMOSメニーコアアーキテクチャでの実現
- まとめと今後の課題、取り組み

サンプル車載制御モデル

連続系と離散系を含む
マルチレート ハイブリッドシステム

コントローラーモデルの各ブロックは、入力のサンプル時間を継承しているため、
***プラントモデル上のシミュレーションでは*、マルチレート ハイブリッド サブシステムとなっている**

（協力）トヨタ自動車、名古屋大学道木研究室

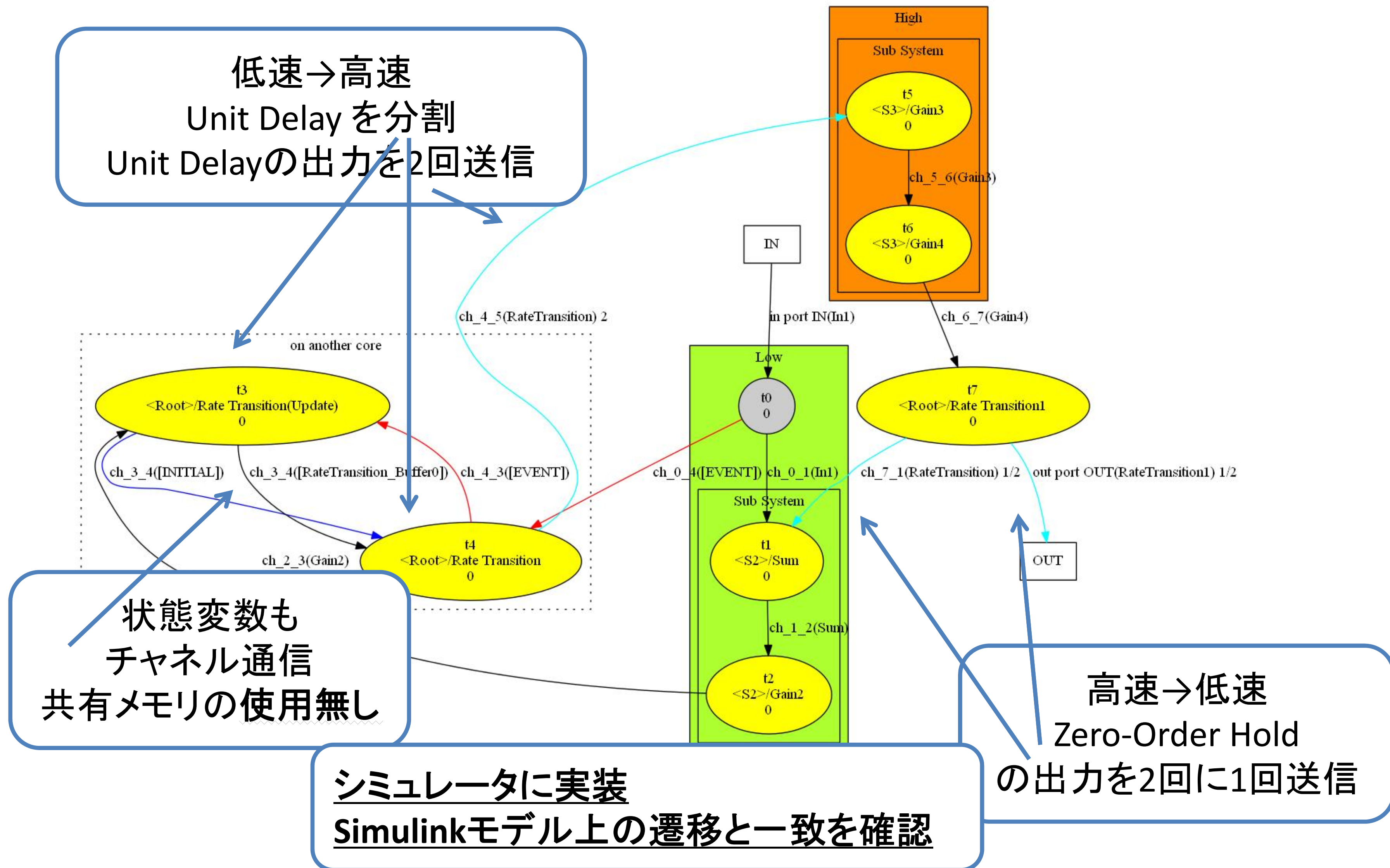
変換フロー

①連続系プラント部分の離散化

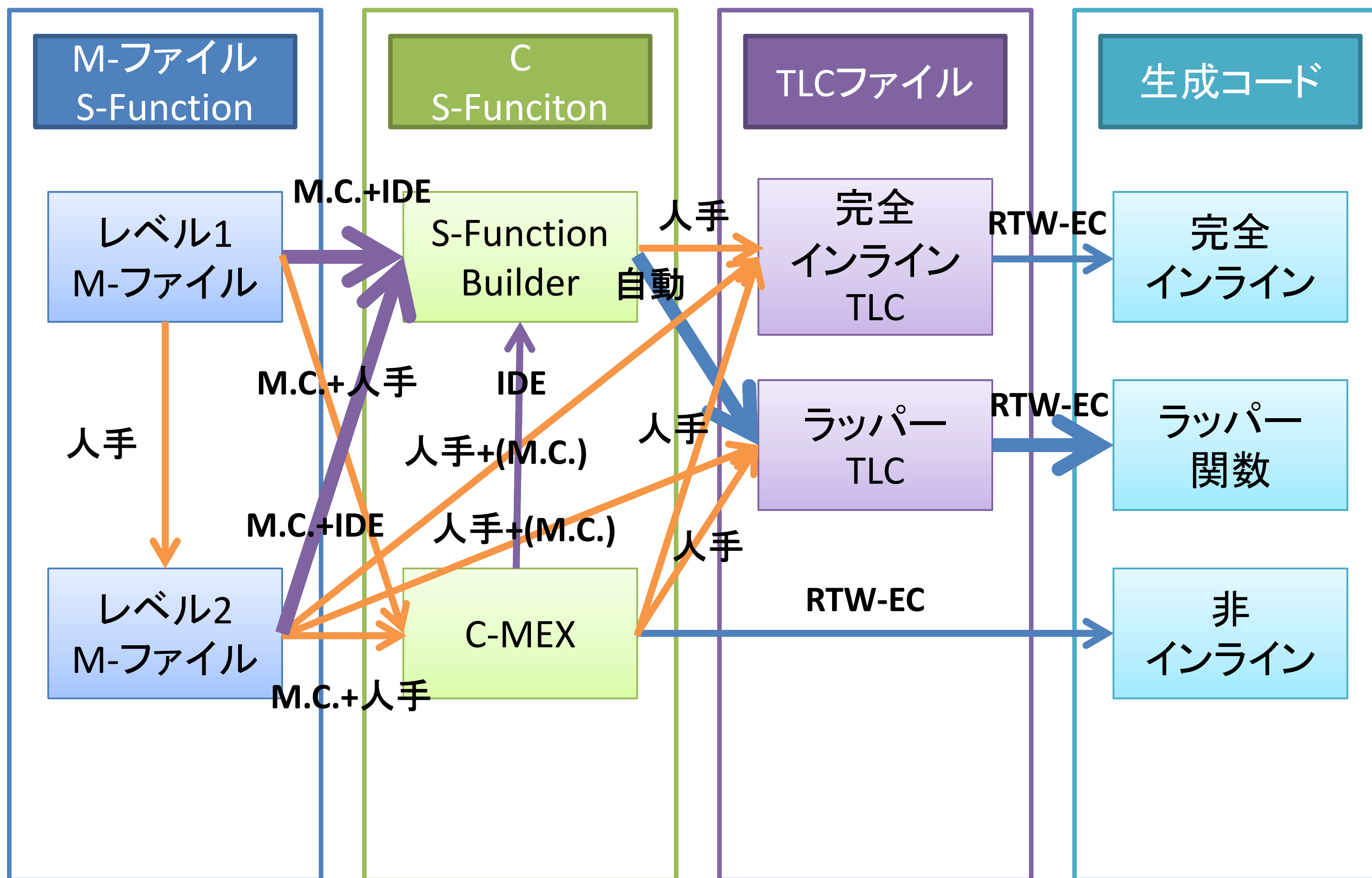
- Mathworksツール上で離散化可能なブロックは離散化
- 連続系のS-Functionを離散化アルゴリズムを用いて、離散系のS-Functionで書き換え
  - その後、同様の方法で、Cに変換
  - 制御周期はControllerの10倍に設定

  - MATLABとCの差分は、6×10^{-14}のオーダー

②マルチレート対応

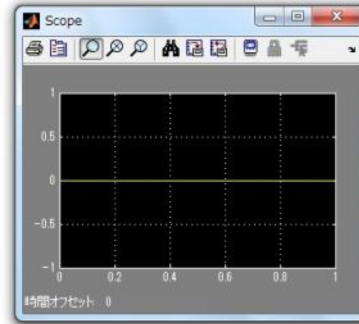


③S-Function からのコード生成フロー (現在のMathworks殿ツール体系に強く依存)

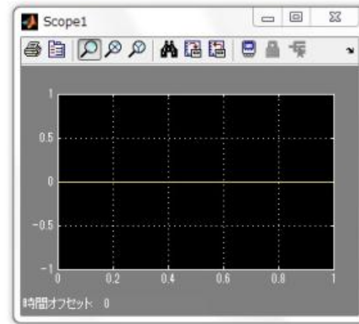


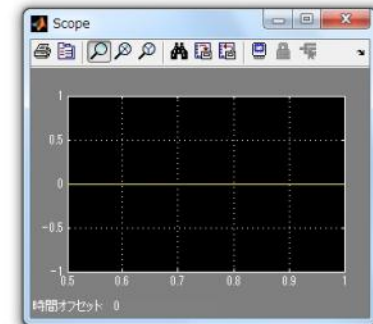
M.C.：MATLAB Coder

③フローにもとづく変換結果 1

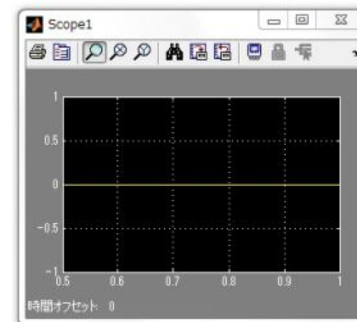


誤差0



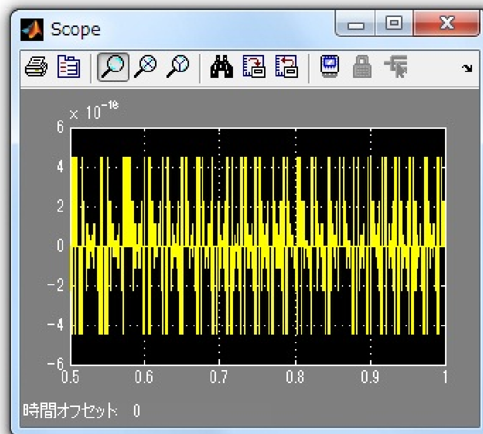


誤差0

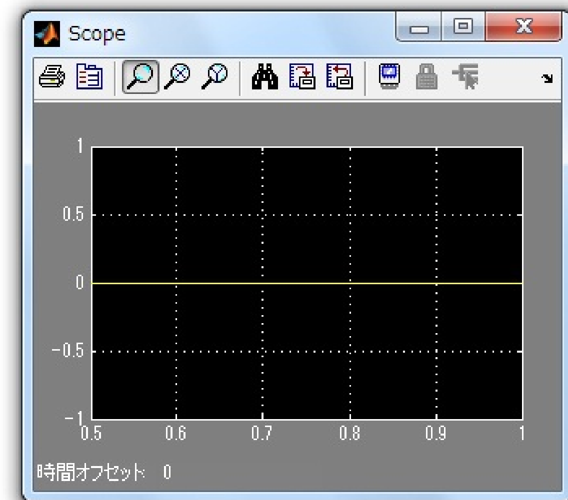


③フローにもとづく変換結果 2

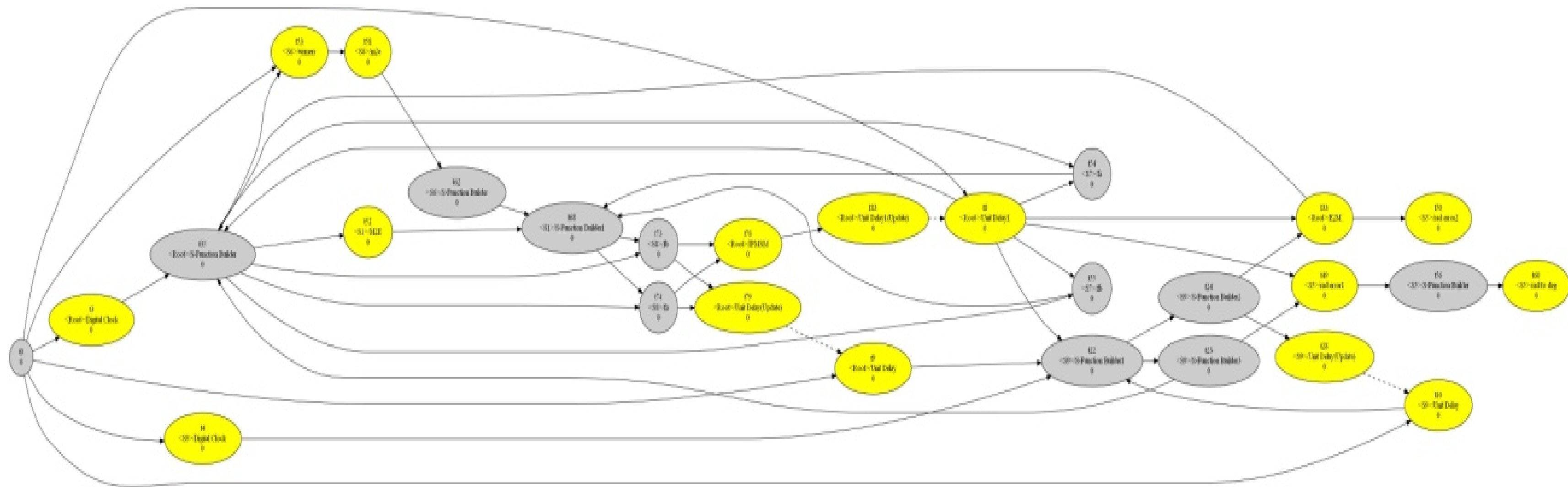
誤差0



C言語の場合、
10^{-16}のオーダーで誤差が生じる
MATLAB言語の場合、-4～4の値を取る

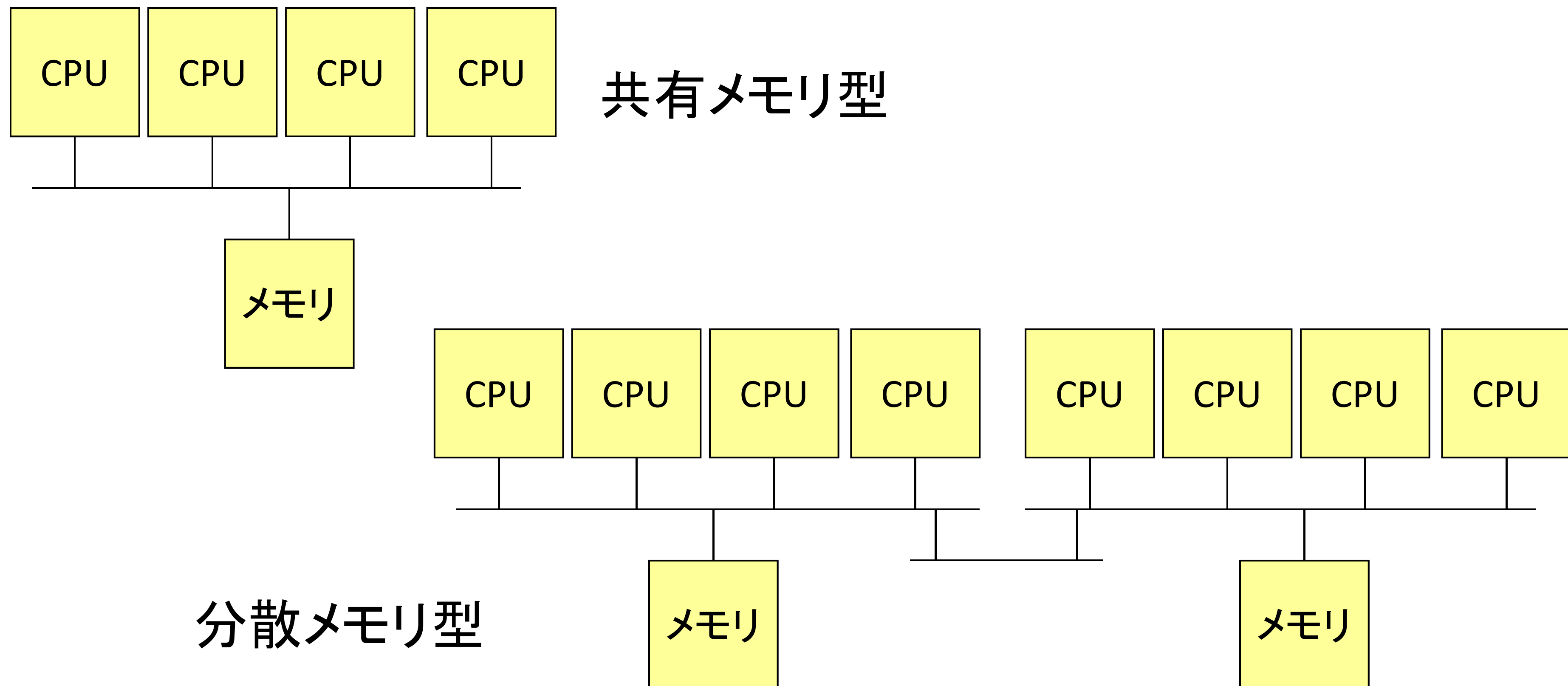


変換結果



④アーキテクチャ依存（１）

- マルチ・メニーコアには様々なアーキテクチャが存在
  - 共有メモリ型マルチコア
  - メニーコアになると分散メモリ化？



④アーキテクチャ依存（２）

- ＯＳモデル、その上の並列モデルに依存
  - 従来のマルチプロセッサの場合
    - 共有メモリならば、OSスレッド (pthread, Windows API)、スレッドライブラリ (OpenMPなど）が主流
    - 分散メモリならば、通信ライブラリ (MPIなど)を利用
    - ライブラリのオーバーヘッドが大きく、それなりの粒度（分割されたプログラム単位の大きさ）が必要

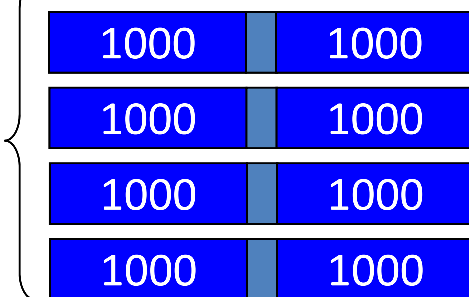
例: 並列オーバーヘッド = 1000 Cycles

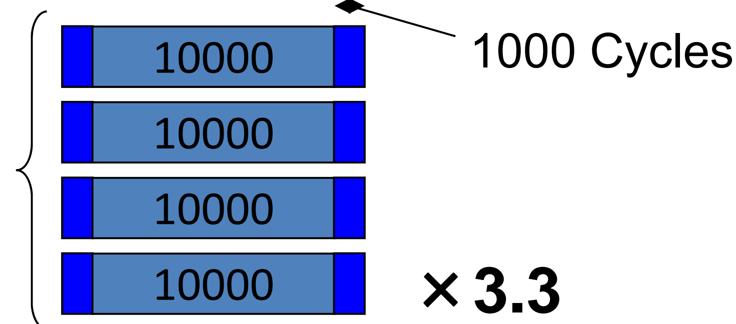
（粒度＝100 Cycles）×４

（粒度=10000 Cycles）×４

1CPU

4CPU
Slower than Before





④アーキテクチャ依存（３）

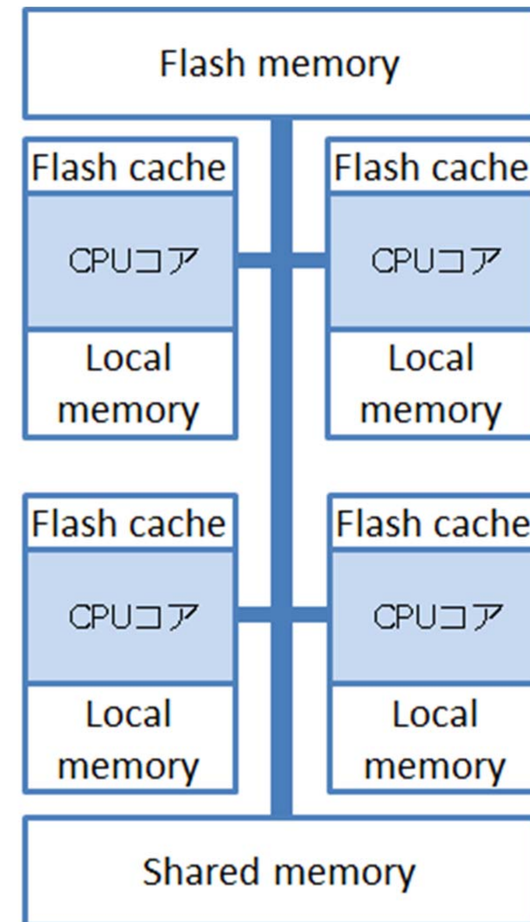
- 車載制御システムの場合、細かく割り込みが入り、その単位での処理になるため粒度が細かい
- 今後の高精度化に向けて、さらに細かくなっていく傾向にある
- 現在のオーバーヘッドは、軽量OSでも数μsecであり、車載制御に適しているとは言えない
- 車載制御システムに適したＯＳ、基本ソフトウェアが必要→今後の大きな研究課題
- マルチコアの例としてV850アーキテクチャ、メニーコアの例としてXMOSアーキテクチャ上で評価した例を示す

アウトライン

- 背景
- MATLAB/Simulinkモデルからの並列化
- 車載制御システムを例とした並列化と課題
  - MATLAB/SimulinkモデルからCSPモデルへ
  - V850マルチコアアーキテクチャでの実現
  - XMOSメニーコアアーキテクチャでの実現
- まとめと今後の課題、取り組み

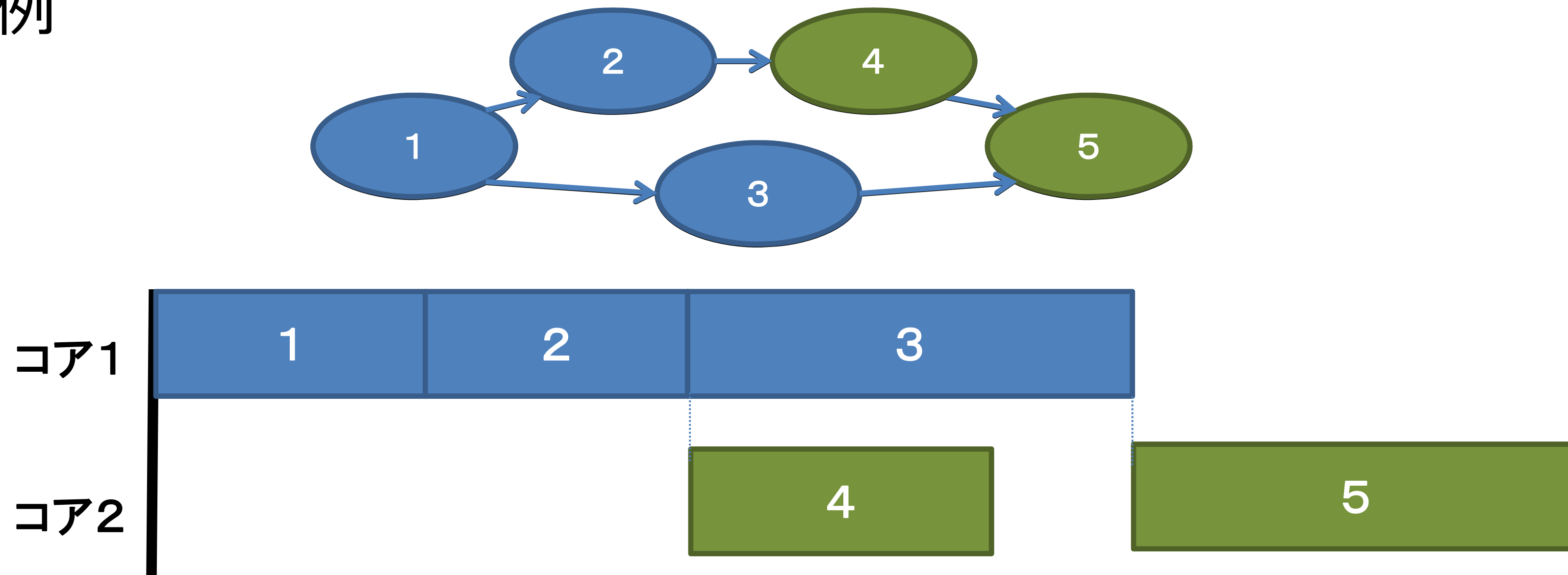
V850マルチコアアーキテクチャ

- 各CPUコア
  - フラッシュメモリ用ノンコヒーレント・キャッシュ
  - ローカルメモリ
- コア間通信
  - 共有メモリ通信
  - 異なるプロセッサ間では通信コストが発生

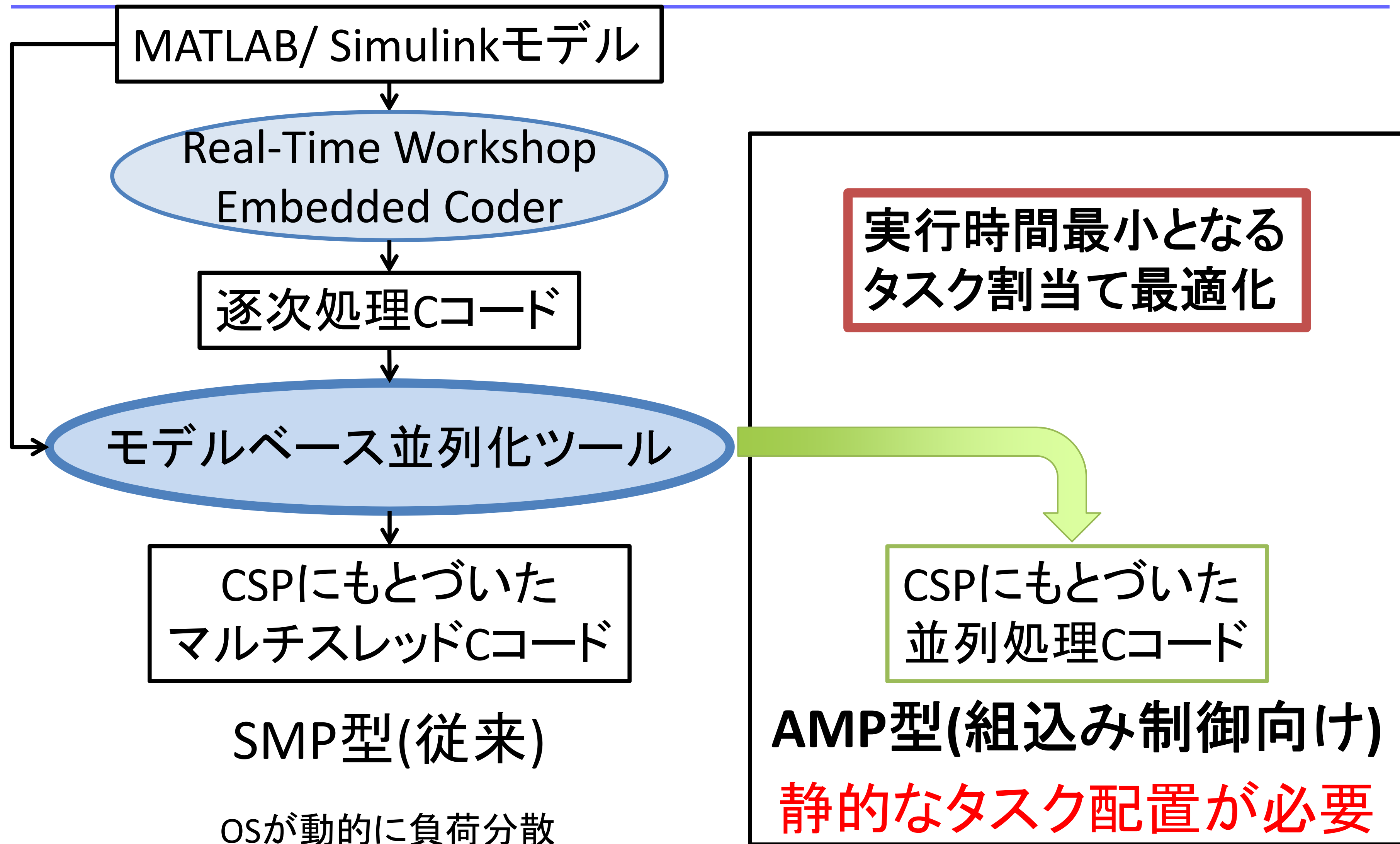


CSPのマルチタスク並列実行

- 各コアで複数のタスクを優先度順に実行
  - 入力モデル全体に制御周期の制約があり、個々のタスクにはデッドライン制約はないと仮定
  - 優先度設計は、入力待ちでのストールが発生しないように、入力側タスクから順に高い優先度を与える
    - タスクフローグラフから自動的に決定
- 例

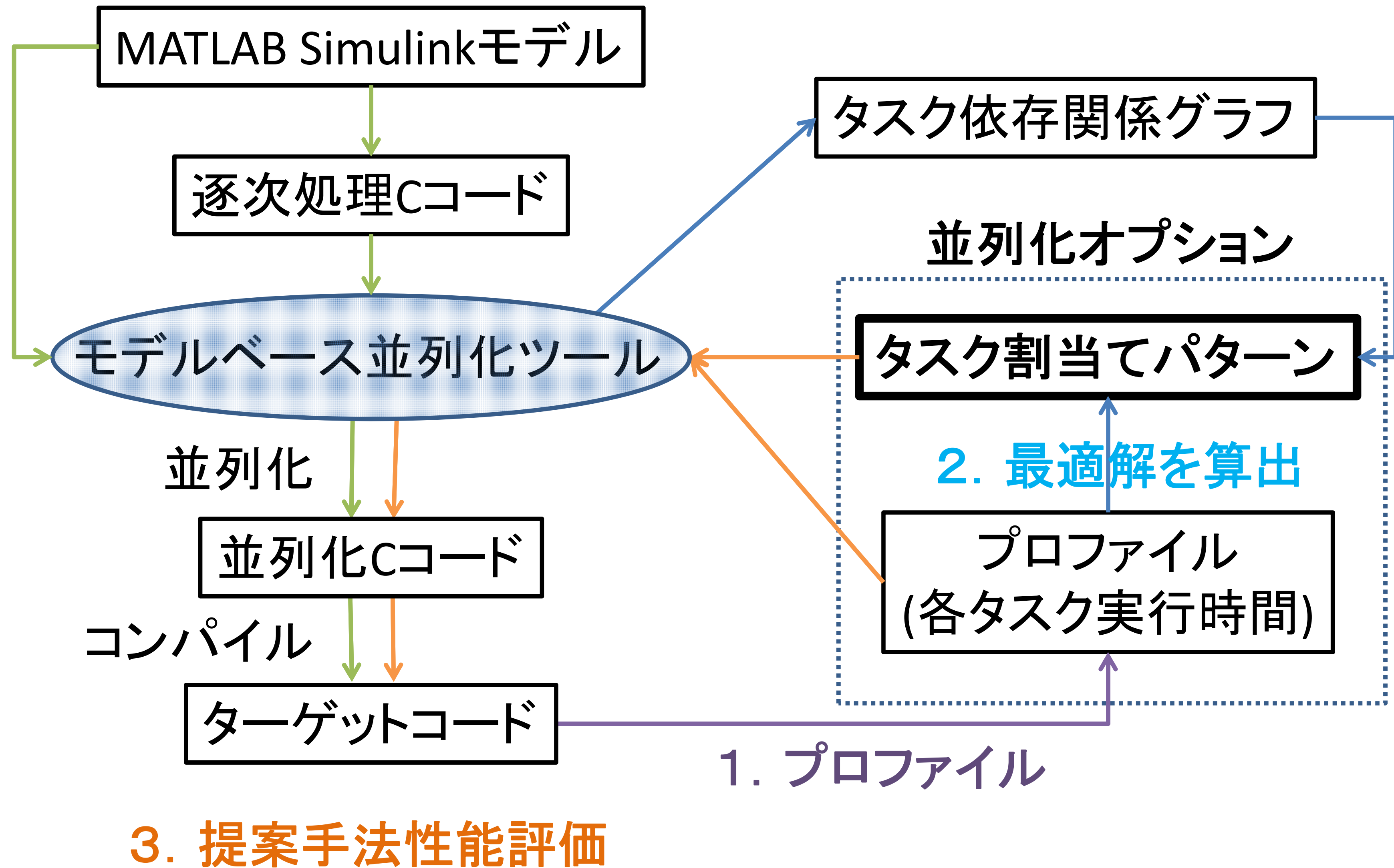


モデルベース並列化ツール



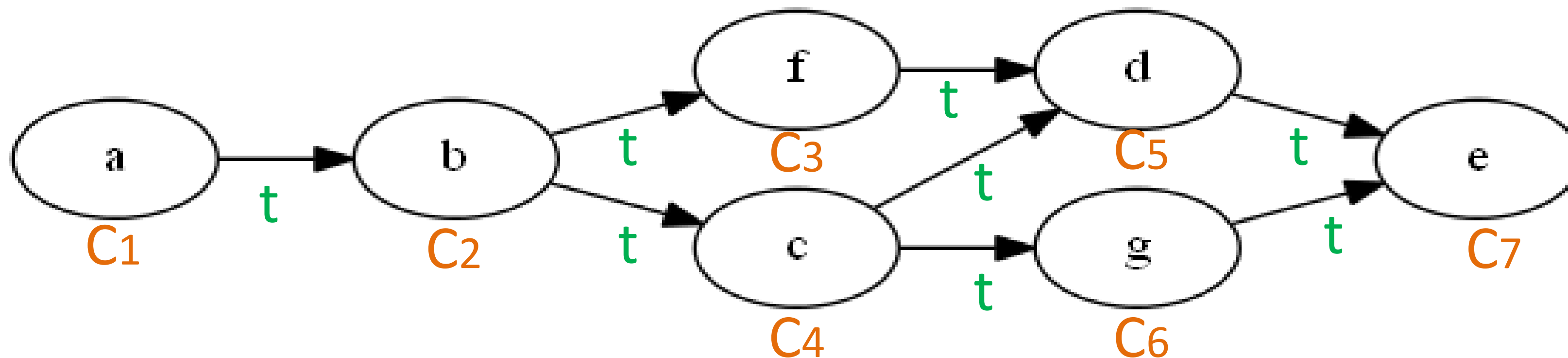
参考文献:久村孝寛. Simulinkモデルにもとづいた並列Cコード生成. 電子情報通信学会技術研究報告, Vol.110, No. 473, pp. 303–308, 2011.

手順



実行時間定式化 (1/4)

- 変数
  - **各タスクの割当てプロセサ** X_i ($1 \leqq i \leqq N$)
- 定数
  - システムのタスク数 N
  - プロセッサ数 M
  - **各タスクの実行時間** C_i ($1 \leqq i \leqq N$)
  - **タスク間依存関係(隣接行列)** A
  - **タスク間通信時間 t**



実行時間定式化 (2/4)

- タスク間通信時間: t
  1. 共有メモリの読み書き: Ciに含まれる
  2. 計算完了のシグナル送信: T (一定)
     - 同プロセッサであれば0

- タスクi - タスクj間の通信時間

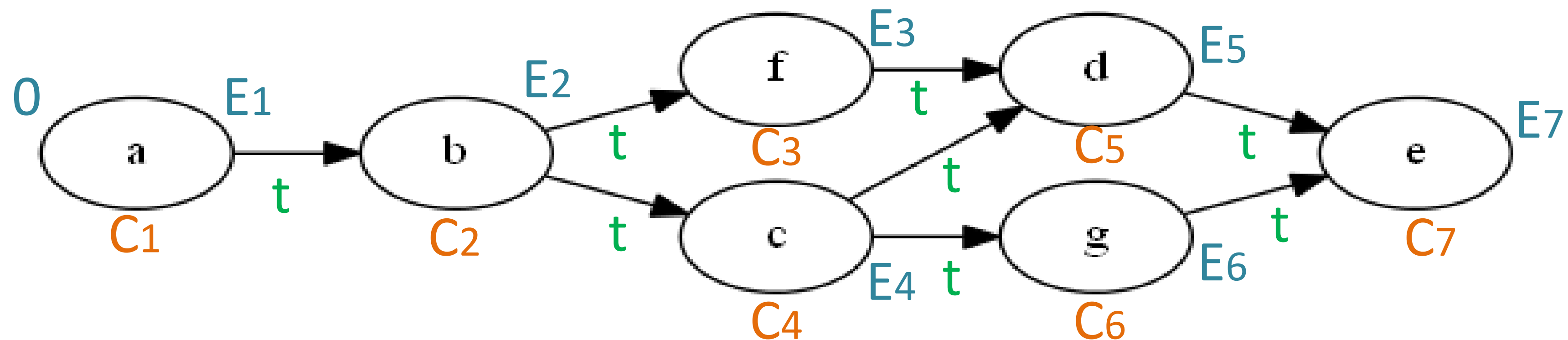
$$t = \begin{cases} T \; (a_{ij} = 1 \text{ and } X_i \neq X_j) \\ 0 \; (\text{otherwise}) \end{cases}$$

依存関係 ($a_{ij} = 1$)

異なるプロセッサ間 ($X_i \neq X_j$)

実行時間定式化 (3/4)

- 先頭タスクの開始を0とし各タスクの完了時間Eiを計算



- モデル実行時間 = **各タスクの完了時間の最大値**
  - $\mathrm{Max}\{E_0, E_1, \cdots E_7\} = E_7$

実行時間定式化 (4/4)

- 目的関数
  - Minimize $\mathrm{Max}\{E_i \mid 0 \leqq i \leqq N\}$
- 制約条件
  - $1 \leqq X_i \leqq M$

- E_i: iの直前に実行されるタスクjの完了時間 + 通信時間
  - iに依存している、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
- $E_i = \max\{E_j + t \mid a_{ji} = 1 \text{ or } (j < i \text{ and } X_i = X_j)\} + C_i$

$$t = \begin{cases} T & (a_{ij} = 1 \text{ and } X_i \neq X_j) \\ 0 & (\text{otherwise}) \end{cases}$$

並列化実験 (1/2)

- モーター制御モデルに対し並列化を行った
  1. コントローラのみ (タスク数 N = 38)
  2. プラント+コントローラ (タスク数 N = 41)

- 評価環境
  - Renesas V850 コアシミュレータ(4コア)
  - プロセッサ数 M = 4
  - 各タスクの実行時間 C_i : プロファイルから取得
  - タスク間依存関係 A : CSPモデルから取得
  - 異プロセッサ間の計算完了のシグナル送信コスト
    - T = 100(サイクル)

並列化実験 (2/2)

- 1stepの実行時間を計測
  - hMETISを用いたグラフ分割手法 (従来手法)
  - 非線形計画問題による最適化 (提案手法)

- １～４コアそれぞれの実行時間比から並列化における性能を評価
  - ソルバーが算出した理論上の実行時間比と比較

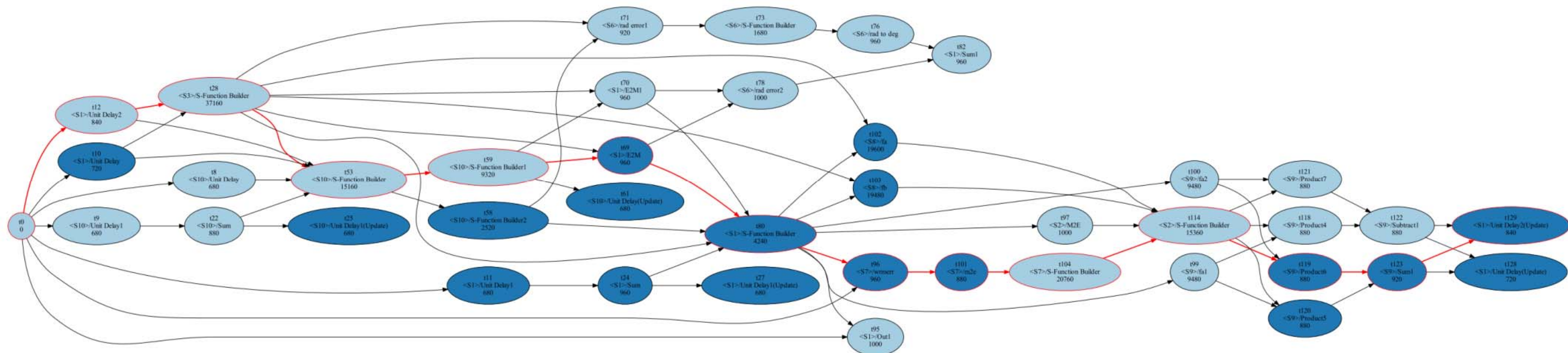
従来手法による割当て

- 同色のタスクは同じコアに割り当てられる
- 赤色で示すエッジはクリティカルパスを示す

- CSPを無向グラフとみなし領域分割
  - 目的関数: 各コアでの実行時間を均等にして実行時間最小化
  - タスク実行順は考慮されない

提案手法による割当て

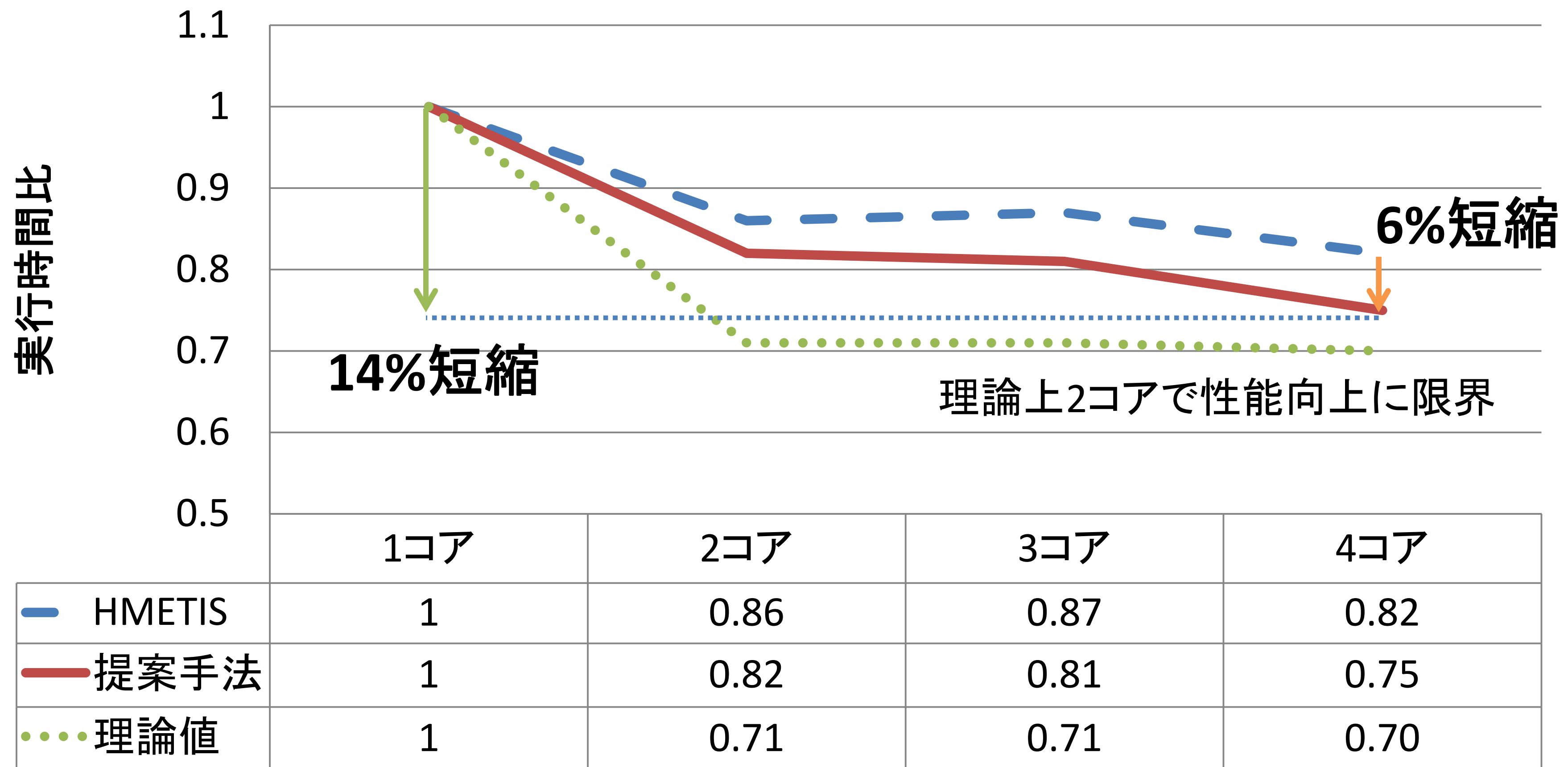
- 同色のタスクは同じコアに割り当てられる
- 赤色で示すエッジはクリティカルパスを示す



- モデル全体の実行時間を定式化
  - プロセッサ空き時間を含むオーバーヘッドを最小化

コントローラ並列化実験結果

- 各コア数における1ステップの実行時間比
  - 逐次実行時の実行時間を1とする

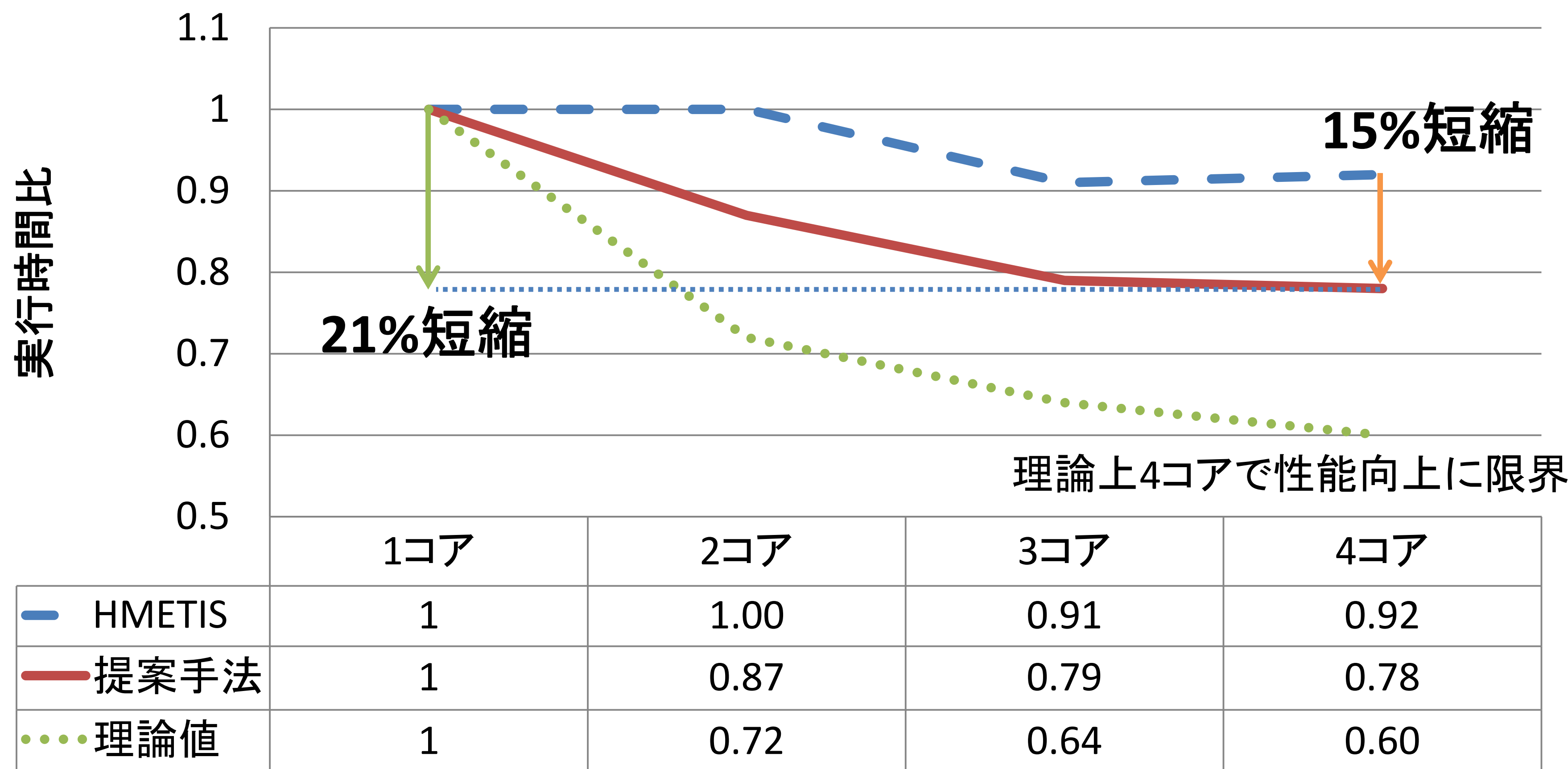


	1コア	2コア	3コア	4コア
HMETIS	1	0.86	0.87	0.82
提案手法	1	0.82	0.81	0.75
理論値	1	0.71	0.71	0.70

逐次実行部分: 67%

プラント・コントローラ並列化実験結果

- 各コア数における1ステップの実行時間比
  - 逐次実行時の実行時間を1とする



	1コア	2コア	3コア	4コア
HMETIS	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

逐次実行部分: 59%

アウトライン

- 背景
- MATLAB/Simulinkモデルからの並列化
- 車載制御システムを例とした並列化と課題
  - MATLAB/SimulinkモデルからCSPモデルへ
  - V850マルチコアアーキテクチャでの実現
  - XMOSメニーコアアーキテクチャでの実現
- まとめと今後の課題、取り組み

何故メニーコア？

- マルチコアでもなかなかスケーラビリティが出ていないのに、どうしてメニーコア実行なのか？

- マルチコア実行の問題点
  - ここでいうマルチコア実行は、「コアあたりのスレッド数が多く、それなりのスレッド管理やスケジューリングが必要なもの」
  - 優先度設計、タイミング設計が必要
  - スレッドライブラリのオーバーヘッドが大きい
    - マルチコア化して、余裕ができたところに新しい制御を入れたいと思ったら余裕はなかった、という話も…

何故メニーコア？

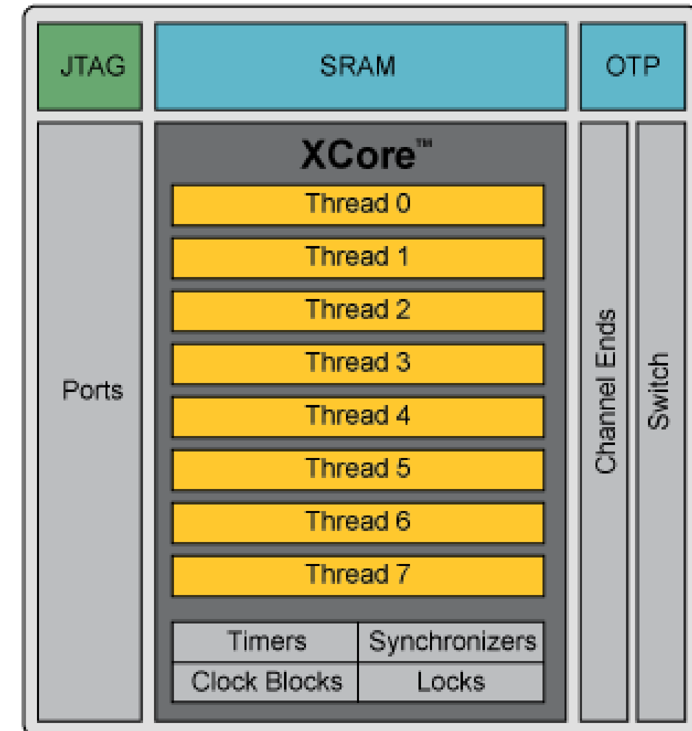
- ＣＳＰモデルのメニーコア実行
  - 優先度設計の必要はない
  - スケジューリングの必要はない

  - 本当に優先度設計、スケジューリングの必要がないことが仮定できるならば、様々なオーバーヘッドを大幅に小さくできる

  - さまざまな可能性が生まれる
    - より複雑な制御を入れる
    - （神経ネットワークのように？）低電力化する
    - メニーコア実行結果を使ってマルチコアに詰め込む方法を考える

XMOSアーキテクチャ

- EDP (Event-driven multi-threaded processor)
  - イベント駆動型プロセッサ(XCORE)によって、イベントドリブンな実行を可能とする。
  - タスクの並列実行をハードウェアレベルで実装
    - １コア８スレッドの同時実行が可能
- "Software Defined Silicon"
  - HDLではなくXCで動作を記述
    - CにOccamの拡張
      (Occam: 並列プログラミング言語)
- CSPモデルの実装に適した
  メニーコアプロセッサ

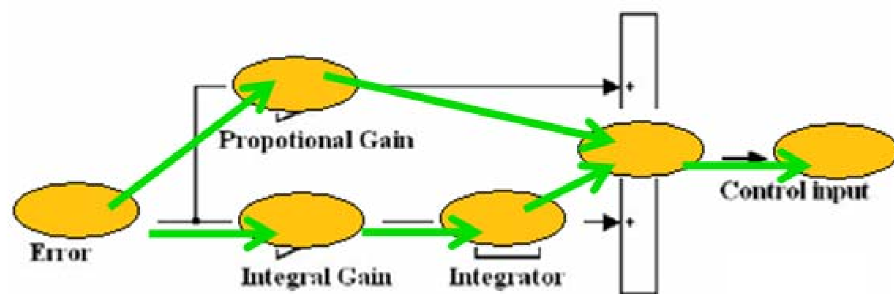


（XMOS社資料より）

CSPモデル→EDP (XMOS)

EDP （Event-Driven Processor）

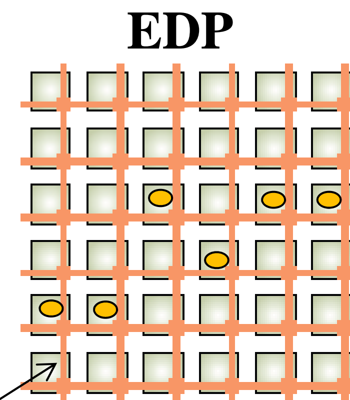
CSPモデルの実装が可能なメニーコア（スレッド）プロセッサ



CSPモデル

実装

EDP



コア(スレッド)

各プロセスを、EDPの各コア（スレッド）に割り当て、実機動作を評価する

XMOSアーキテクチャ

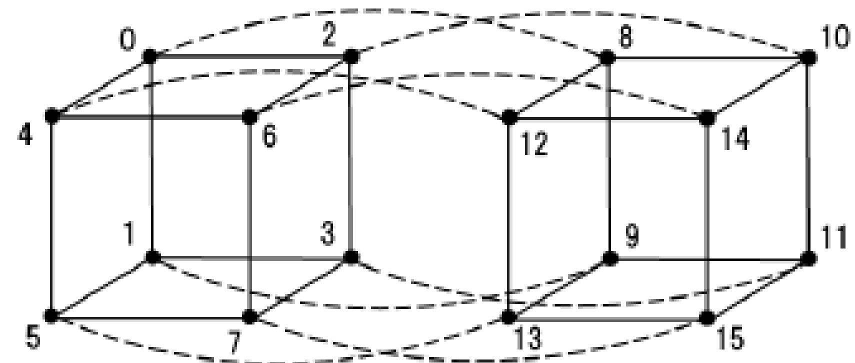
- CSPの「プロセス」をXMOSの「スレッド」に固定的に割付、かつ「プロセス」は「スレッド」を専有
- XMOS：コアあたり８スレッド、ハードウェア実装で、時分割処理
- 通信はスレッド間 nsecオーダー

⇒固定＆専有割付のため、スレッドのスケジューリングは不要、通信も高速でオーバーヘッドが小さい

⇒車載制御系でも性能向上が見込める

XK-XMP64

- 1チップ4コア16チップ
  - 計64コア
- チップのトポロジーはHypercube構造
  - コア間距離は0～3ホップ
  - コア間距離に応じて通信時間が線形的に増加



モデル実行時間計測

- 1ステップの実行時間評価
  - 10回の計測の平均値

手法	実行時間(μsec)
スレッド並列 (Intel Core i7 6コア 3.33GHz)	90.8
XMOS実行 (64コア 400MHz、シミュレータでの測定）	92.7

- （参考）XMOSの電力 450μW/MHz・コア (*)
  - フル稼働としても約1W ← $(450*10^{-6})*400*64$

(*) XMOS社 2009年6月24日プレスリリースより

アウトライン

- 背景
- MATLAB/Simulinkモデルからの並列化
- 車載制御システムを例とした並列化と課題
- まとめと今後の課題、取り組み

まとめと今後の課題、取り組み

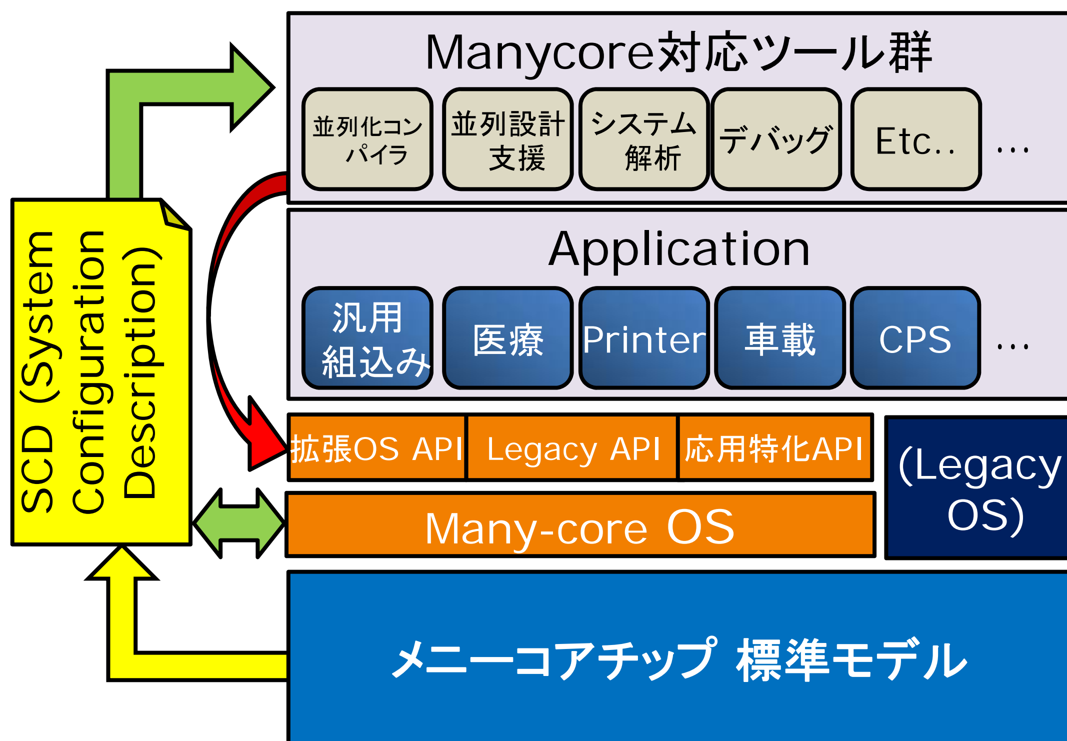
- 車載制御システムの高性能化
- それを支えるマイコンの省電力実行のためには
マルチ・メニーコア化が必須
- たくさんのコアを有効に使うため、
ソフトウェアの並列化が必要不可欠
- モデルベースからの並列化と、車載制御システムを用いた評価を紹介
- 今後の課題
  – モデルベースからの並列化のみでは解決しない
  – 制御、実装の協調、歩み寄りが必要
  – 開発・実行環境全体のプラットフォーム構築が必要

制御・実装の協調、歩み寄り

- 制御からの歩み寄り
  - 並列化容易な制御系アルゴリズム
  - 早い段階からの離散系への変換
  - 依存の少ない制御モデル
    - 例：次の周期に間に合えばよいデータの明記
- 実装からの歩み寄り
  - オーバーヘッドの少ない基本ソフト
    - メニーコアＯＳ
  - メニーコアを利用したハードウェア支援並列化
    - 例：分岐両実行、投機的実行
  - アーキテクチャ依存の解消
  - マルチ・メニーコア向けツール体系、プラットフォーム化

メニーコアチップ/ツール/OSフレームワーク

- OSを始め、各種開発支援ツール、異なるメニーコアチップを利用できるフレームワークが必要、メニーコアチップ標準モデルはその基盤となる



メニーコアチップ標準モデルを策定 (実装並びに拡張は各ベンダの競争領域)

高いリアルタイム性と、ヘテロなメニーコアを透過的に扱えるスケーラブルなメニーコアOS(メニーコアチップ標準モデルに適合させる)

既存ソフトウェアの再利用性確保

高性能な新規ソフトのための開発環境

安心・安全のための高信頼性