Vol.2012-SLDM-155 No.22
Vol.2012-EMB-24 No.22
2012/3/3

JISX0180:2011 「組込みソフトウェア向けコーディング規約の作成方法」を用いた Parallelizable C の定義

木村啓二[†1] 間瀬正啓[†1,†2] 笠原博徳[†1]

組込みソフトウェアの品質向上を目的として，JISX0180:2011「組込みソフトウェア向けコーディング規約の作成方法」が策定された．一方，自動並列化コンパイラによる並列性抽出を補助するための Paralleliza C が提案されている．本稿では，組込みソフトウェア開発者の自動並列化コンパイラ活用によるマルチコア用アプリケーション開発の生産性向上を目的とし，JISX0180:2011 による Parallelizable C の定義を提案する．本コーディング規約によるプログラムを商用 SMP 及び情報家電用マルチコア上で評価した結果，8 コアの IBM p5 550Q では平均 5.54 倍，4 コアの Intel Core i7 960 では平均 2.43 倍，4 コアの Renesas/Hitachi/Waseda RP2 では平均 2.79 倍の速度向上をそれぞれ得ることができた．

A Definition of Parallelizable C by JISX0180:2011 “Framework of establishing coding guidelines for embedded system development”

Keiji Kimura,[†1] Masayoshi Mase[†1,†2]
and Hironori Kasahara[†1]

JISX0180:2011 “Framework of establishing coding guidelines for embedded system development” was decided to improve the quality of embeded systems. Parallelizable C has bee also proposed to support exploitation of parallelism by a parallelizing compiler. This paper proposes a definition of Parallelizable C by JISX0180:2011 aiming at the improvement of productivity for embeded multicore developers with parallelizing compilers. An evaluation has been carried out using rewritten programs by the defined coding guideline on ordinary SMPs and a consumer electronics multicore. As the result, 5.54x speedup on IBM p5 550Q (8core), 2.42x speedup on Intel Core i7 960 (4core), and 2.79x speedup on Renesas/Hitachi/Waseda RP2 (4core) have been achieved, respectively.

1. はじめに

組込み機器の身の回りの様々な用途への利用拡大に応じて組込みソフトウェアの複雑さと規模も増大しており，ソフトウェア生産性や保守性の向上，及び品質の向上が大きな問題となっている．この問題を，特に C 言語によるソフトウェアのコーディング作法の規定により解決することを目的として，ヨーロッパの自動車業界を中心に MISRA-C が策定された[1)]．

また日本では，上記 MISRA-C を参考にした上で，開発するソフトウェアのソースコードの書き方をプロジェクトあるいは組織単位で統一的に定め，これをコーディング規約として規定することによってソフトウェアの品質を維持しようとする目的の下，JISX0180:2011「組込みソフトウェア向けコーディング規約の作成方法」が策定された[2)]．

本 JIS 規格は，ソフトウェアのソースコードの品質を保つためのコーディング作法を，信頼性，保守性，移植性，及び効率性の観点から体系化して提示している．さらに，規格の附属書では C 言語用の具体的なルールを提示している．プロジェクト責任者は，本規格に沿ってプロジェクトあるいは組織のための具体的なソースコードの規約を策定及び運用することとなる．また本 JIS 規格は，この規約作成時にプロジェクトあるいは組織の要件に合わせ，ルールの修正や追加が可能なことを特徴とする．

一方，マルチコアプロセッサはサーバーや PC はもちろん，組込み機器においても利用されるようになっているが，未だにそのソフトウェア開発の難易度は高い．逐次プログラムの自動並列化コンパイラによる並列化は，これらマルチコアプロセッサ用ソフトウェアの生産性を大きく向上する技術として注目されている．しかしながら，特に C 言語はその記述の自由度の高さから，アルゴリズムそのものに並列性が存在しても，自動並列化コンパイラによる十分な並列性が抽出できない場合がある．

上記のような問題に対して，IMEC からマルチプロセッサ向けのコーディングガイドラインである Clean C が提案されている[3)]．さらに，並列性抽出の際に特に問題となるコンパイラのポインタ解析の精度に着目し，自動並列化コンパイラが並列性を抽出しやすいコーディング作法を定めた Parallelizable C が提案されている[4)]．

†1 早稲田大学
Waseda University
†2 現在 日立製作所勤務
Currently Hitachi, Ltd.

Vol.2012-SLDM-155 No.22
Vol.2012-EMB-24 No.22
2012/3/3

本稿では，マルチコアを用いた組込み機器における並列ソフトウェアの生産性や保守性等のび品質の向上を目的とし，Parallelizable C の JISX0180 による定義を提案する．

以下，第 2 節では JISX0180 の概要を，第 3 節では Parallelizable C の概要をそれぞれ述べる．第 4 節では Parallelizable C を JISX0180 により再定義し，第 5 節で Parallelizable C で記述されたプログラムを SMP サーバや情報家電用組込みマルチコアで評価した結果を述べる．

2. JISX0180:2011 組込みソフトウェア向けコーディング規約の作成方法

JISX0180:2011「組込みソフトウェア向けコーディング規約の作成方法」は，組込みソフトウェアの品質向上を目的とし，コーディング規約の定義並びに運用の方法を規定したメタルールである．

本 JIS 規格においては，コーディング作法とはソースコードの品質を保つための慣習及び実装の考え方であり，作法概要と作法詳細から構成される．これらのコーディング作法を具体化したものがコーディングルールとして定義される．そして，これらコーディングルールを整理することにより，組織あるいはプロジェクトに応じたコーディング規約が定義される．

本 JIS 規格では，コーディング作法及びコーディングルールを，信頼性，保守性，移植性，及び効率性といった品質特性の観点から整理している．各品質特性には略号がつけられており，R が信頼性，M が保守性，P が移植性，E が移植性をそれぞれ表す．さらに，各コーディング作法概要，コーディング詳細，及びコーディングルールは上記略号と階層的な番号付けで識別される．例えば，R1 は信頼性に分類される「領域は初期化し，大きさに気をつけて使用する」という作法概要を，R1.1 はその作法概要に属する「領域は初期化してから使用する」という作法詳細を，そして R1.1.1 は作法詳細 R1.1 に属し，C 言語の場合には「自動変数は宣言時に初期化するか，又は値を使用する直前に初期値を代入する」というコーディングルールをそれぞれ表す．

これらコーディング作法，コーディング規約，コーディングルール，及び品質特性の四分類の関係を図 1 に示す．

図中，例えばコーディング作法中の作法概要における「修正し誤りの無いような書き方にする」といった項目は，信頼性及び保守性の品質特性に関連しているが，これらの品質特性の中で特に関連深い保守性に分類されていることを実線のエッジで示している．さらにこの項目を詳細化した作法詳細の項目として「ブロックは明確化し，省略しない」を定めており，これを具体化したコーディングルールとして「if, else if, else, while, do, for, switch 文

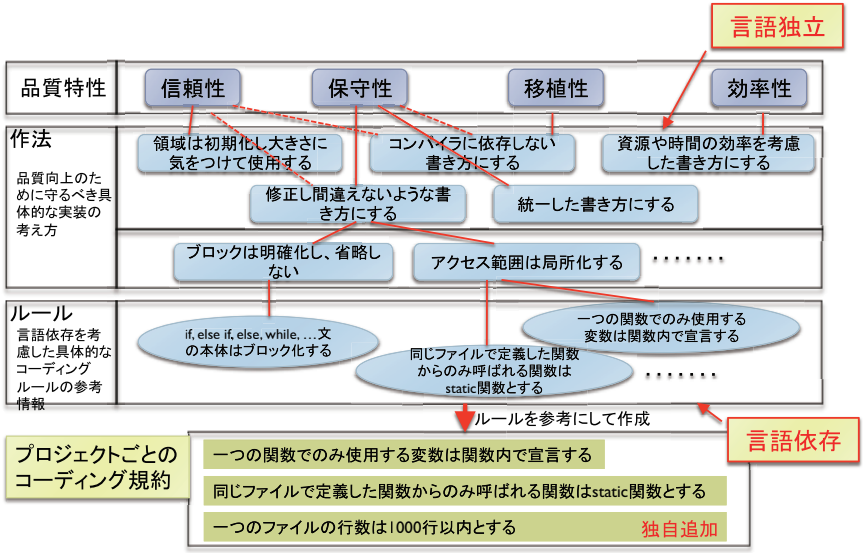

図 1 品質特性，コーディング作法，コーディングルール，及びコーディング規約の関係
Fig. 1 The relationship among quality characteristics, coding conventions, coding rules, and a coding guideline

の本体はブロック化する」といった項目を定めている．ここで，コーディング作法は言語独立（一部依存），コーディングルールは言語依存となる．本 JIS 規格では，C 言語に対しては附属書の形式でコーディング規約を示している．

個々の組織，あるいはプロジェクトに対しては，これらのコーディングルールから目的に沿ったものを取捨選択し，場合によっては独自にルールを追加することにより，コーディング規約を定義する．作法やルールの修正や追加を行った場合は，その理由の明示が推奨されている．図 1 では，「一つのファイルの行数は 1000 行以内とする」が独自追加部分となる．

以上のように定義されたコーディング規約は，レビューにより検証及び妥当性の確認を行う．さらに，定期的なレビューによってプロジェクトや組織の変化に応じたコーディング規約の継続的な改善が推奨されている．

本 JIS 規格は，作成したコーディング規約の運用方法に関しても規定している．この運用方法には，作成したコーディング規約が実情にそぐわなかった場合の適用除外に関するルール，コーディング規約の教育，及びコーディング規約の充実，といったものが含まれている．

Vol.2012-SLDM-155 No.22
Vol.2012-EMB-24 No.22
2012/3/3

3. Parallelizable C

Parallelizable C は自動並列化コンパイラによる並列性抽出を補助するためのコーディングガイドラインである．並列性抽出のためにどのようなルールが必要となるかは，自動並列化コンパイラの並列性抽出能力に依存する．C 言語の場合，この並列性抽出を行う基盤となる情報として，ポインタ解析の結果を用いることになり，このポインタ解析の精度が並列性抽出能力にとりわけ強い影響を与える．文献 4) で提案された Parallelizable C では，以下のようなポインタ解析精度，及び構造体の解析精度を想定している．

- ステートメントごとにコントロールフローに沿った解析を行う（flow-sensitive）[5]
- 関数の呼び出し箇所ごとに解析を行う（context-sensitive）[5]
- メモリ動的確保までの関数呼び出し経路ごとにヒープオブジェクトを区別する（heap-cloning）[6]
- 構造体のメンバをそれぞれ別のオブジェクトとして扱う（field-sensitive）[7]
- ポインタがオブジェクトの先頭を指すかどうかを判別する
- ポインタの配列について各要素の差し先に重なりがないかを判別する[8]
- ヒープオブジェクトを確保したイタレーションを判別する[8]
- スカラとして扱う構造体のメンバは別のオブジェクトとして扱う
- 配列として扱う構造体はそれ自体を配列オブジェクトとして扱い，配列要素内のメンバの区別は行わない

上記解析精度に基づいて規定された Paralallelizable C のコーディングルール，及びその詳細と規定理由は以下のようになる．なお，以下のルールを逸脱したソースコード上にあっても，単にそのコード部分に対して並列性抽出のための解析を行わないだけでありコンパイルエラーとはならない．

1. 変数宣言の境界を越えたアクセスを行わない

構造体のメンバや多次元配列の各次元の境界を越えたアクセスを行わない．言語仕様上未定義であり，ポインタ解析器は本ルールを逸脱したアクセスを想定しない．本ルールは以降のルールの前提となるルールである．

2. ポインタのキャストを行わない

メモリ動的確保時を除いて，ポインタのキャストを行わない．ポインタ参照先オブジェクトのオフセットを正しく解析するためである．

3. ポインタ算術演算を行わず添え字アクセスを行う

ポインタ算術演算は行わない．ポインタ差し先の領域にアクセスする場合はポインタ変数の値は更新せずに，ポインタ変数に対する逆算賞演算子（*）もしくは添え字アクセス（[ ]）により行う．

4. 条件分岐やループ内でポインタ変数への値の代入を行わない

メモリ動的確保時を除いて，条件分岐やループ内ではポインタ変数への値の代入を行わない．ポインタの指し先を一意に定めるためである．

5. 同一領域に対する複数のポインタを関数の引数として渡さない

配列の単一次元内の異なるオフセットのアドレスを関数の引数として渡さない．ただし，同一配列内でも重なりのない部分配列であれば良い．これは，2 つの引数の差し先を別々のオブジェクトとして扱えなくなるためである．

6. 一時バッファとして利用するヒープ領域を使い回さない

ヒープ領域の使い回しは，通常の変数の使い回しよりも解析が困難であり，並列性抽出の阻害要因となる．

7. 構造体の配列メンバに対する配列アクセスを行わない

配列メンバを持つ構造体の配列を使用せず，構造体のメンバに配列がある場合は，下位の関数に構造体の 1 要素に当たる領域を引数として渡し，下位の関数内でそのメンバの配列へのアクセスを行う．これは，配列要素間と同様に，構造体メンバ間の識別精度も解析精度に影響するためである．

8. 再帰的なデータ構造を利用しない

リストや木構造などの再帰的なデータ構造の解析とそれを用いた処理の並列化には，それぞれのデータ構造に対する専用の特殊な解析と特殊な並列実行モデルが必要となるためである．

9. 呼び出し間に依存のある外部関数を使用しない

ファイル入出力，エラー処理等の，ライブラリ内で内部状態を持ち，呼び出し間の順序関係に制約のある外部関数を使用しない．エラー処理やログ出力は並列化したい部分の外側にくくり出す等する．

10. 関数の再帰呼び出しを使用しない

関数は直接的あるいは間接的に関わらず，その関数自体を呼び出さない．静的なコールグラフ決定の困難さ，及びメモリ使用量見積もりの困難さのためである．

11. 関数ポインタを使用しない

関数ポインタは使用しない．これは関数の呼び出し先の静的な解析が困難なためである．

Vol.2012-SLDM-155 No.22
Vol.2012-EMB-24 No.22
2012/3/3

12. 可変個引数を持つ関数を定義しない

可変個引数の使用はその関数で定義・参照される変数の解析が困難なため使用しない.

4. JISX0180:2011 を用いた Parallelizable C の再定義

4.1 基本方針

JISX0180 を用いた Parallelizable C の再定義の基本方針として，すでに本 JIS 規格に類似のコーディングルールがある場合はこれを積極的に利用する．特に，信頼性の高いコーディングルールに沿って記述されたプログラムは，その挙動が決定的かつ限定的であり，自動並列化コンパイラによるプログラム解析（特にポインタ解析）が行いやすいため,「信頼性」に分類されるコーディングルールからいくつかを選択する.

4.2 定　　義

以下に，上記基本方針に基づき JISX0180 を用いて定義した Parallelizable C のコーディング規約を提示する.

まず，JISX0180 の附属書 A から選択したコーディングルールを示し，各ルールに対して 3 節で示した文献 4) 中のコーディングルールのうち対応するものを併記する.

- R1.3.1(1) ポインタへの整数の加減算（++ 及び −− も含む.）は使用せず，確保した領域への参照・代入は [ ] を用いる配列形式にする
  - （対応）1. 変数宣言の境界を越えたアクセスを行わない
  - （対応）3. ポインタ算術演算を行わず添え字アクセスを行う
- R2.7.1(1) ポインタ型は，他のポインタ型，及び整数型に変換しない．また，逆もしない．ただし，データへのポインタ型における void*型との変換は除く
  - （対応）2. ポインタのキャストを行わない
- R2.8.2(1) 可変個引数をもつ関数を定義しない
  - （対応）12. 可変個引数を持つ関数を定義しない
- R3.4.1（修正） 関数は，直接的か間接摘果に関わらず，その関数自体を呼び出さない
  - （対応）10. 関数の再帰呼び出しを使用しない

次に，Parallelizable C の要件を満たすコーディング規約を規定するために，JISX0180 に追加するコーディングルールを以下に示す．これらのコーディングルールは四つの品質特性のうち，特に「効率」に関連する.

- 4. 条件分岐やループ内でポインタ変数への値の代入を行わない
- 5. 同一領域に対する複数のポインタを関数の引数として渡さない
- 6. 一時バッファとして利用するヒープ領域を使い回さない
- 7. 構造体の配列メンバに対する配列アクセスを行わない
- 8. 再帰的なデータ構造を利用しない
- 9. 呼び出し間に依存のある外部関数を使用しない
- 11. 関数ポインタを使用しない

自動並列化コンパイラを利用してマルチコア用ソフトウェアを C 言語で記述する場合，そのプロジェクトあるいは組織では，上記 4 つのコーディングルールを JISX0180 の附属書 A から選択し，7 つのコーディングルールを新規に追加した上で，必要に応じてコーディングルールを選択・修正・追加しコーディング規約を策定することになる.

5. 評　　価

本節では，Parallelizable C による逐次 C プログラムの記述，並びにそれらプログラムの自動並列化コンパイラによる並列性能評価について述べる[4].

本評価で用いたアプリケーションは SPEC CPU 2000 より art, qeuake，SPEC CPU 2006 より lbm, hmmer，MediaBench より mpeg2encode，株式会社ルネサステクノロジご提供の AAC エンコーダである．当初から Parallelizable C で記述されている AAC エンコーダ以外のプログラムを必要に応じて Parallelizable C のコーディング規約を満たすように書き換え，OSCAR マルチグレイン自動並列化コンパイラ[9] を用いて並列化した．本評価で用いた OSCAR マルチグレイン自動並列化コンパイラに実装されているポインタ解析器は，3 節で述べた解析精度を満たしている．評価プログラムは OSCAR コンパイラにより並列化され，OSCAR API[10] 入り C プログラムとして出力される．並列化されたプログラムから OpenMP コンパイラあるいは API 解釈系 + 逐次コンパイラにより実行バイナリが生成される.

各プログラムの Parallelizable C への書き換えを表 1 にまとめる．art と equake はオリジナルのソースコードが Parallelizable C の規約に準拠しているため，書き換える必要が無かった．ただし，equake においてはプログラム実行の大部分を占めるループにおける配列リダクション処理のためのリストラクチャリングを手動で適用した．lbm と hmmer は表中に示すように Parallelizable C に準拠していない部分を修正したが，その書き換え量は lbm で 1.8%，hmmer で 0.03%である．mpeg2encode の書き換え量が 23.5%と多いが，これは MPEG2 エンコード処理に本来存在するマクロブロックレベルの並列性を抽出可能とするための書き換えとなっている.

Vol.2012-SLDM-155 No.22
Vol.2012-EMB-24 No.22
2012/3/3

表 1 Parallelizable C への書き換え
Table 1 Summary of rewrite into Parallelizable C

プログラム	合計 (LOC)	書き換え項目	削除量 (LOC)	追加量 (LOC)	変更率 (%)
SPEC2000 art	1270	N/A	0	0	0%
SPEC200 equake	1513	N/A	0	0	0%
SPEC2006 lbm	1155	オブジェクト先頭以外へのポインタ ループ内のポインタ更新	-7 -10	+7 +14	1.8%
SPEC2006 hmmer	35992	ヒープ領域の再利用	-9	+8	0.03%
MediaBench mpeg2encode	3750	アルゴリズム変更	-640	+864	23.5%

表 2 評価環境
Table 2 Evaluation Environment

System	IBM p5 550Q	Intel Core i7 920 PC	Renesas/Hitachi/Waseda RP2
CPU	Power5 + (1.5GHz x 2 x 4)	Core i7 920 (2.66GHz x 4)	SH-4A (600MHz x 4)
L1 D-Cache	32KB/core	32KB/core	16KB/core
L1 I-Cache	64KB/core	32KB/core	16KB/core
L2 Cache	1.9MB/2core	256KB/core	
L3 Cache	36MB/2core	8MB/4core	
バックエンド コンパイラ	IBM XL C/C++ Version 10.1	Intel C/C++ Version 11.0	SH C Compiler + 専用 API 解釈系

並列性能評価には，8 コア搭載 SMP サーバである IBM p5 550Q，4 コア搭載 Intel Core i7 の PC，及びルネサステクノロジ，日立製作所，早稲田大学で開発した 8 コア搭載情報家電用マルチコア RP2（ただし本評価では 4 コア SMP モードを使用）[11] を用いた．表 2 に各評価環境の諸元を示す．

上記プログラムを IBM p5 550Q, Intel Core i7 920 PC, Renesas/Hitachi/Waseda RP2 上で並列性能評価をした結果を図 2，図 3，図 4 にそれぞれ示す．各図は，縦軸がオリジナルソースの逐次実行に対する速度向上率を表し，横軸が評価アプリケーションとその各々に対するオリジナルソース（SPEC2000， SPEC2006，あるいは MediaBench）と書き換え後のソース（Parallelizable C）の 1 コア（1core），2 コア（2core），4 コア（4core），8 コア（8core）での実行結果をそれぞれ表す．IBM p5 550Q における equake の「-O4」は，他のプログラムはバックエンドコンパイラである IBM XL コンパイラの最高位最適化オプション「-O5」でコンパイルしたが，XL コンパイラの不具合により equake だけ「-O4」で最適化したことを示している．

オリジナルソースから Parallelizable C の規約に沿って書き換えた lbm，hmmer，及び mpeg2encode に着目すると，図 2 より，IBM p5 550Q 8 コア使用時では各アプリケーションのオリジナルソースで逐次実行に対して，lbm と hmmer は速度向上をせず，mpeg2encode では 1.53 倍の速度向上率であったものが，Parallelizable C 準拠に書き換えることにより lbm で 5.35 倍，hmmer で 6.06 倍，mpeg2encode で 5.12 倍の速度向上がそれぞれ得られたことがわかる．

同様に図 3 より，Intel Core i7 920 PC 4 コア使用時では，lbm と hmmer では速度向上が得られず，mpeg2ncode で 1.81 倍の速度向上率であったものが，書き換えにより lbm で 1.28 倍，hmmer で 3.34 倍，mpeg2encode で 3.60 倍の性能向上がそれぞれ得られたことがわかる．また，図 4 より，Renesas/Hitachi/Waseda RP2 4 コア使用時では，lbm と hmmer では速度向上が得られず，mpeg2encode で 1.61 倍の速度向上率であったものが，書き換えにより lbm で 2.19 倍，hmmer で 3.47 倍，mpeg2encode で 3.27 倍の速度向上率がそれぞれ得られたことがわかる．

art, equake, AACencode も含めると，8 コアの IBM p5 550Q では平均 5.54 倍，4 コアの Intel Core i7 960 では平均 2.43 倍，4 コアの Renesas/Hitachi/Waseda RP2 では平均 2.79 倍の速度向上をそれぞれ得ることができた．

以上より，Parallelizable C のコーディング規約により，オリジナルのソースコード，あるいは比較的少ない書き換え量で自動並列化コンパイラにより並列性抽出効果が得られることが確認できた．

6. ま と め

本稿では，組込みソフトウェアの品質向上を目的とし，かつプロジェクトや組織ごとの状況に合わせて柔軟にコーディング規約の柔軟な規定を可能とする JIS 規格である，JISX0180:2011「組込みソフトウェア向けコーディング規約の作成方法」を用いた，自動並列化コンパイラによる並列性抽出を補助するコーディング規約である Parallelizable C の再定義を提案した．

再定義した Parallelizable C は JISX0180 の附属書 A から選択した 4 つのコーディングルールと，新規に追加した 7 つのコーディングルールから構成される．

Parallelizable C で SPEC CPU 2000，SPEC CPU 2006，MediaBench のプログラム，及び商用利用されている AACencode プログラムを必要に応じて修正し，OSCAR コンパイラで並列化したものの並列性能評価を行ったところ，わずかな書き換え量で 8 コアの

Vol.2012-SLDM-155 No.22
Vol.2012-EMB-24 No.22
2012/3/3

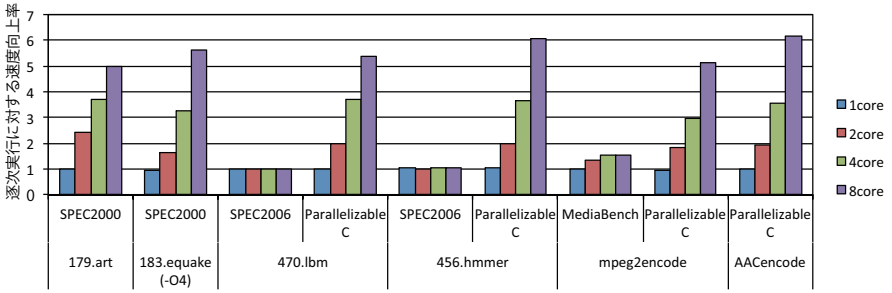

図 2 IBM p5 550Q における並列性能評価
Fig. 2 Scalability evaluation result on IBM p5 550Q

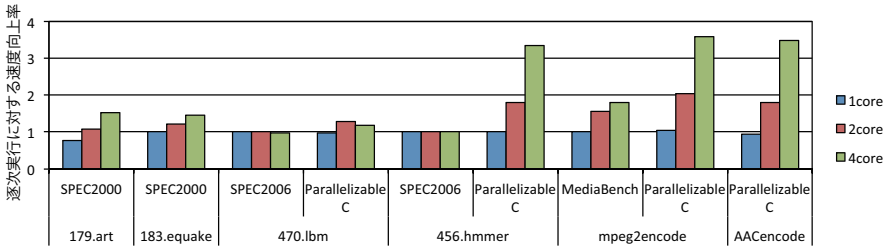

図 3 Intel Core i7 920 PC における並列性能評価
Fig. 3 Scalability evaluation result on Intel Core i7 920 PC

IBM p5 550Q では平均 5.54 倍，4 コアの Intel Core i7 960 では平均 2.43 倍，4 コアの Renesas/Hitachi/Waseda RP2 では平均 2.79 倍の速度向上をそれぞれ得ることができた．

本稿で提案した JISX0180 による Parallelizable C を各プロジェクトや組織で利用することにより，組込みソフトウェアの品質向上に加え，マルチコア用ソフトウェアの生産性向上及び性能向上を期待することができる．

謝辞 本研究の一部は経済産業省 “グリーンコンピューティングシステム研究開発” の支援により行われた．また，「組込みソフトウェア向けコーディング規約の作成方法 JIS 原案作成委員会」の皆様に感謝いたします．

参 考 文 献

1) *Guidelines for the use of the C language in critical systems* (2004).

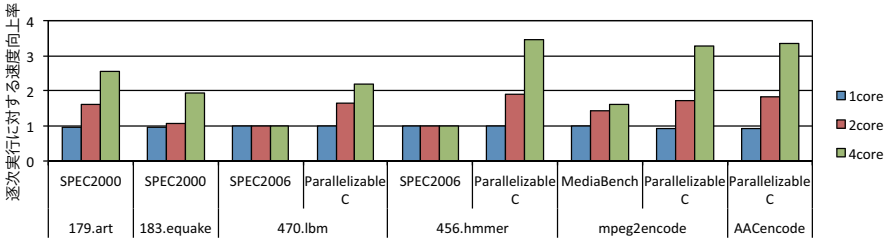

図 4 Renesas/Hitachi/Waseda RP2 における並列性能評価
Fig. 4 Scalability evaluation result on Renesas/Hitachi/Waseda RP2

2) 日本規格協会：JISX0180:2011 組込みソフトウェア向けコーディング規約の作成作法 (2011).
3) IMEC: Clearn C, , available from ⟨http://www.imec.be/CleanC⟩ (accessed 2012-01-24).
4) Mase, M. et al.: Parallelizable C and Its Performance on Low Power High Performance Multicore Processors, *15th Workshop on ompilers for Parallel Computing* (2010).
5) Emami, M. et al.: Context-sensitive Interprocedural points-to analysis in the presence of function pointers, *Proceedings of the ACM SIGPLAN 1994 conference on Programming Language Design and Implementation*, pp.242–256 (1994).
6) Nystorm, E.M. et al.: Importance of Heap Specialization in Pointer Analysis, *Proceedings of the 5th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pp.43–48 (2004).
7) Pearce, D.J. et al.: Efficient field-sensitive pointer analysis for C, *Proceedings of the 5th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and enginnering*, pp.37–42.
8) 間瀬正啓ほか：自動並列化のための Element-Sensitive ポインタ解析，情報処理学会論文誌プログラミング（PRO），Vol.3, No.2, pp.36–47 (2010).
9) Kasahara, H. et al.: A Multi-grain Parallelizing Compilation Scheme for OSCAR (Optimzally Scheduled Advanced Multiprocessor), *Proc. 4th Intl. Workshop on LCPC*, pp.283–297 (1991).
10) Kimura, K. et al.: OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers, *Lecture Notes in Computer Science*, Vol.5898, pp.188–202 (2010).
11) Ito, M. et al.: An 8640 MIPS SoC with Independent Poweroff Control of 8 CPUs and 8 RAMs by an Automatic Parallelizing Compiler, *Proc. ISSCC2008* (2008).