組込みソフトウェアの動態分析に基づく性能改善手法

外山　正勝　　黒澤　寿好　　松本　利夫
三菱電機（株）　情報技術総合研究所

1. はじめに

近年，携帯電話を代表とする高機能な組込み機器の開発では，ソフトウェアの開発規模が急激に拡大しており，特に機能欠陥に対する品質確保のための取り組みが中心的に行われてきてきた。その結果，標準プラットフォーム化という考え方が広まり，規模の拡大に対して破綻することなくソフトウェアを動作させる環境が整備されつつある。

しかしながら，性能品質に対しては，取り組みが十分ではない部分がある。例えば，あるユーザ操作に対する応答速度は，許容範囲が人それぞれ異なるために厳密な基準を設定することが難しく，厳しい開発工程の中では対策が後回しにされてしまうことが多々ある。一方で，コンシューマからは組込み製品の高機能化に伴う応答速度の低下を指摘する声が挙がっており，速度性能面での品質も重要視されている。

本稿では，このような性能面の品質問題に対し，効率良く改善する手法について検討する。

2. 組込みにおける性能改善の取組み

2.1 組込み製品開発のおかれた状況

組込み製品で使われるハードウェアは，小型，省電力，低コストなどの要求が厳しく，性能的な余裕は必要最小限となるように設計されている場合が殆どである。従って，組込みソフトウェアのプログラミングでは，ハードウェアの能力を最大限に引き出すために最適化された実行コードが要求されている。

一方で，ソフトウェアの開発規模が拡大してきたことにより，設計段階から性能を考慮することが難しくなってきている。開発スタイルとして，プラットフォーム化や分業化が進められたことにより，システム試験のような開発終盤のフェーズで性能問題が指摘されるケースが多くなっている。これは，システムが複雑化するにつれ，機能の組合せや競合パターンが急増するために，従来のような全ての組合せにおいて設計段階で事前に性能を検証することが難しくなってきたためである。

また，上記に対してさらに，開発工期の短縮という時間的制約が加わる。開発期間が限られた状況においては，終盤フェーズでの性能問題に対して，ソフトウェア構造の見直しを伴うような根本的な対策は，将来に向けての課題として先送りされることが多い。そのため，構造変更を伴わない部分的な性能チューニングによって，許容される水準まで性能を改善するような取り組みがなされている。

2.2 性能改善の手段

開発終盤フェーズにおける性能改善の進め方としては，目標性能値を設定し，次のような作業が行われる。これらの作業は，粒度や範囲を変更しながら目標性能の達成に向けて繰り返し施行される。

① 動的解析による対象の絞り込み
　プロファイリングツールや実行時ログ情報などを用いて，性能問題が指摘された機能に対応するプログラムをターゲット機器上で動作させることで，実行単位や処理単位毎の実行時間を把握する。そして，ボトルネックとなっている箇所をリストアップする。

② 処理内容の把握
　リストアップされた箇所のソースコードの記述内容を解析することにより，プログラムの構造や参照関係を把握し，実行効率が悪いと判断される処理を人手で見つけ出す。その際，構造把握を支援するために，静的解析ツールが用いられることもある。

③ コードの最適化
　選別した箇所のコードを，より実行効率のよいものへと改修する。例えば，冗長な処理の削減，CPU のパイプライン処理やキャッシュ利用率の効率化，ウェイトの調整など，部分的な処理の最適化を行う。

④ 効果の確認
　再度動的解析を行い，改善効果を確認するとともに，次の改善対象を選定する。

A method of performance improvement for embedded software with dynamic and static analysis parameters
Masakatsu Toyama, Hisayoshi Kurosawa, Toshio Matsumoto
Information Technology R&D Center, Mitsubishi Electric Corp.

2.3 性能改善における課題

2.2 節で示した方法によって性能改善を進めていくにあたり，次の課題が挙げられる。

- 時間がかかっている処理を特定したとしても，一定量の改善効果が必ずしも得られるわけではない。
- 時間を多くかければそれだけ効果が得られる可能性は高まるが，効率は低下していく。

性能チューニングを効率的に行うためには，改善にかかるコストを考慮しながら，期限内に最大限の効果が得られるよう，対象を絞り込むことが重要がある。しかし，実際には，効果や効率は，人の技術レベルや経験・ノウハウといった面に強く依存してしまっている。

3. 提案手法

3.1 目的

本手法では，工期の限られた開発における終盤フェーズにおいて，性能改善を効果的・効率的に行うことを目的としている。そのために，性能改善作業において，人的要因に依存している，効果的な改善対象の抽出部分について自動化する手段を検討する。

3.2 手段

本手法では，動的解析によって得られたソフトウェアの動態情報と，静的解析による構造情報とを組み合わせて分析することにより，費用対効果が最大となる性能改善候補を抽出する。

分析対象は関数を単位とし，ある関数 f の性能改善にかかるコストC_fと改善効果E_fの 2 つの評価関数を考える。このとき，改善効率は$E_f \div C_f$で表される。この改善効率に従って関数をランク付けすることにより，費用対効果の大きな性能改善候補を抽出することができる。

ただし，大きな効果を得るためにはコストをかける必要があり，また改善率の向上に伴いコストあたりの効果は低下していくと予想されるため，コストと効果には統計的に図 1 のような相関関係があると考えられる。従って，トレードオフとなる基準を設定して各々の関数を評価する必要がある。

コストについては，コード上にチューニングによる工夫の余地がどの程度含まれているかによって予測が可能である。これは，関数の構造に大きく依存している。従って，コストを求める評価関数は，関数の構造情報をパラメータとして与えることによって定義することができる。構造パラメータとしては，関数規模，分岐経路数，ループ規模，ネスト深度，メモリコピー回数等が挙げられる。例えば，関数規模が大きければ，工夫の余地が生まれやすいため，コストは低くなる。なお，パラメータ間で干渉するものもあり，例えば，経路数が多くなるほど，実質的な関数規模は低く見積もる必要がある。

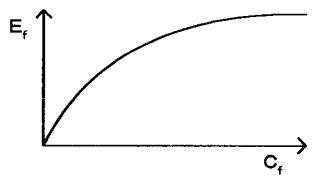

図 1　コストと効果の関係

次に，改善効果について述べる。即効性のある改善効果を得るためには，動的解析による動態情報を考慮すべきである。動態情報として，関数の実行時間や呼出し回数，ループ回数，実行経路等が挙げられる。例えば，呼出し回数が多ければ，小さな改善が大きな効果につながる。

3.3 関数規模によるコストの評価

3.2 節に示したモデルをもとに，最も単純なパターンである関数規模 L [f]のみを構造パラメータとして用いる場合について考察する。動態情報として，関数毎の実行時間 T [f]を用いる。

各々の関数 f に対して，目標性能値として実行時間をχ%改善する場合のコストC_fを，チューニングの難易度という観点から導出し評価を行う。このとき，改善効果E_fは$T[f] \times \chi \div 100$であり，動態情報からその効果を確認できる。

改善率が一定であれば，L [f]に対するC_fの関係は図 2 のようになると予想される。これは，同程度の性能改善を達成するのであれば，規模が大きい方が工夫の余地が生まれやすいため，性能チューニングの難易度は下がることを表している。これに基づき，性能改善コストの低い関数の自動抽出が可能である。

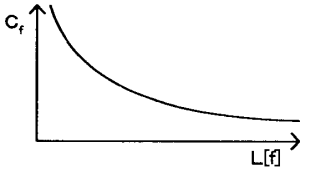

図 2　規模とコストの関係

4. おわりに

性能改善を行う手法として，ソフトウェアの動態情報と構造情報とを組み合わせ，コストと効果を分析することにより，効率よく効果が得られる性能改善対象を抽出する手法を示した。

今後は，実際のソフトウェア開発に適用し，この方式の検証を行っていく。また，複数のパラメータを追加した複雑なケースでの評価方法ついても考察を進めていく。