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

2004－MBL－31　(2)
2004－ＩＴＳ－19　(2)
2004／11／11

組込み SW の特性に基づいたバージョン間差分抽出方式

清原良三[†α]　栗原まり子[†]　三井 聡[†]　古宮章裕[‡]

最近, 携帯電話をはじめとする組込み機器の機能が増大の一途をたどっている．これにつれて, 機能のカスタマイズや, 更新, 不具合の修正など機能の更新に関する要求が高まっている．携帯電話においては開発期間, 評価期間の関係から不具合が発生する可能性が高くなり, 不具合修正のためのネットワークサービスを利用する例もでてきている．

そこで, 旧版のソフトウェアと新版のソフトウェアの差分のみを送信しするためのバイナリ差分抽出の技術が重要となっている．本論文では, バイナリ差分の抽出に関して携帯電話の構造に着目して効果的に差分量を減らす方法を提案する．提案した方法を携帯電話のソフトウェアに適用し, 評価を実施し, 効果があることを示す．

A New Method of differencing between two SW versions

Ryozo Kiyohara[†α], Mariko Kurihara[†], Satoshi Mii[†],Akihiro Komiya[‡]

Recently, due to increasing functions in embedded systems (mobile phones and etc.), it requires customizability, updating , bug fix and etc. Especially mobile phones, it is difficult to release bug-free software because of short periods of developments and evaluations. Some network services for bug fix are available for mobile phones.

That case, it is important to calculate binary delta between old version and new one. In this paper, we discuss about a new binary delta method for mobile phones, evaluate that method and show good performance about delta size.

1　はじめに

最近, 携帯電話をはじめとする組込み機器の機能が増大の一途をたどっている．これにつれて, 機能のカスタマイズや, 更新, 不具合の修正など機能の更新に関する要求が高まっている． 携帯電話においては開発期間, 評価期間の関係から不具合が発生する可能性が高くなり, 不具合修正のためのネットワークサービス [1] を利用する例もでてきている．

携帯電話などの組込み機器は, コストの関係でメモリを極力少なくしており, 組み込んだソフトウェアはそのまま ROM 上で実行できる形式とし,RAM にロードする必要がない構成にしていることが多い． 即ち ROM 上にアドレス解決済みでコードを置くため, 少量の不具合修正であっても参照先のアドレスがずれることにより, 更新が必要となる箇所が多数発生し, 少しの修正でも予想外の大きさの修正を余儀なくされる場合がある．

また, 組込み機器特に携帯電話のソフトウェアを更新する場合は, 電池の問題などにおいて更新中に失敗する可能性を減らすために更新時間を小さくするべきである．更新時間のほどんどはは以下に示す 2 種類の時間からなる．

[†]三菱電機 (株) 情報技術総合研究所
[†]Mitsubishi Electric Corppration
[‡]三菱電機インフォメーションシステムズ (株)
[‡]Mitsubishi Electric Information Systems
[α]kiyohara@isl.melco.co.jp

- 更新のための修正情報を転送する時間
- フラッシュ ROM を書き換える時間

我々は, 更新のための修正情報を転送する時間を小さくする方法を [2] で提案している．さらにフラッシュ ROM 上のデータの書き替えを小さくする方法に関しては [3] で提案しており, いずれも評価の結果では効果があることを示している．

更新のための修正情報を転送する時間は, 新版と旧版のバイナリレベルでの差分データサイズに依存する． このバイナリ差分の表現方法に関して, さらに組込みソフトウェアの差分の現われ方の特徴を利用することにより, より小さく表現できないかを検討し, 命令コードのアドレス部の修正量が多い点に着目した差分データ圧縮方式を提案し, 評価した．その結果, 従来の方式に比べ有効であることがわかった．

2 関連研究

ソフトウェアの差分による更新技術は, 差分情報の抽出と差分を小さく表現する技術および転送量を減らすための圧縮技術からなる．

ソフトウェアの差分抽出方法に関しては,UNIX の diff [4][5] が代表的なアルゴリズムである． diff のアルゴリズムはテキストファイルを対象にしており, 出力された差分を適用することも想定したコマンド列を出力することもできる．しかし, 対象がバイナリファイルの場合はそのままでは使えない．

このようなアルゴリズムで, バイナリファイルに対して適用可能にした例として bdiff[6], や suff[7] がある．こららは, 組込み機器に適用するように想定したものではなく, 実行時間や, メモリの利用量, 差分の表現サイズなどに関して課題が残る． 文献 [8] では, 差分圧縮という考え方で, 圧縮同様の方法で差分を作成するアプローチを提案し, 効果的な方法であることを示している．

静的な差分抽出ではないが, ネットワークで結ばれた端末間で SW の差分を動的に取得することにより, 差分更新をする rsync [9] が提案されており, バージョン管理をしていなくても効果的に更新可能である．

また, 差分を表現する方法としては gdiff[10] が W3C にて提案されている．差分を適用するためのコマンドを COPY と DATA の 2 つ主として定義して, この 2 つで差分を表現する． COPY コマンドでは, コピーするサイズを指定する必要があるが, そのサイズや位置を指定するビット長によってコマンドを変えるというような工夫がされている．

また, この gdiff をさらに発展させ, このコマンドテーブルを実際のデータにあったように可変にするというような工夫を VCDIFF[11] では行うことにより差分を小さく表現できるとしている． さらに [11] では, 差分圧縮の考え方を利用して, 新たに書き換えたデータをも利用するという工夫もされている．

1 行の修正により, コンパイルしたときのレジスタアサインのずれが起こり, 修正した場所の周辺に影響を及ぼす場合も考えられる．このようなケースには, 機械語では一定のずれが周辺の命令（レジスタを使う命令）で起こるものと想定され, オフセットの数値を使うことにより, 差分が小さく表現できることも報告されている [12]． また, 修正はアドレス部に中心に起こるものとし, アドレス部には関係なく差分を表現し, アドレス部のずれは別途新版上で計算するという手法も提案されている．

次に余分なメモリスペースの少ない組込み機器上で有効にメモリを利用しながら書き換えを行うアルゴリズムとして [13] も提案されている．この方法によれば、ワークメモリがほとんど不要になるという効果がある．

いずれの方法も汎用的な差分抽出および表現方法を目指している．一方, 我々が目的とする組込みソフトウェアの差分更新では差分の現われ方に一定の特性がある．本研究では, この特性を利用した差分表現形式を提案する．

3 差分のパターンの観察

大規模な組込みソフトウェアとして携帯電話を例にとり検討をした． 対象とした携帯電話は, CPU として M32R[14] を採用しておりμ iTRON 上で動作する． プログラムはすべてアドレス解決をしたリンク済みのコードとしてフラッシュ ROM 上に格納されている．

このようなケースにおいては， 1 行の追加や削除で，全体の位置がずれると，図 1,2に示すように

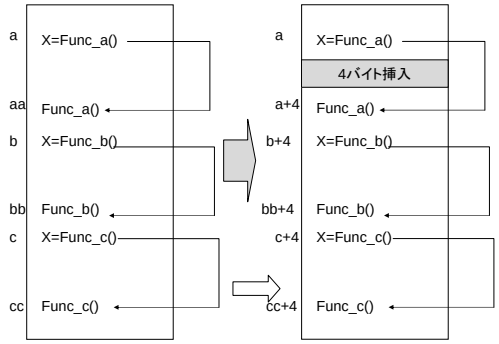

図 1: 修正の影響

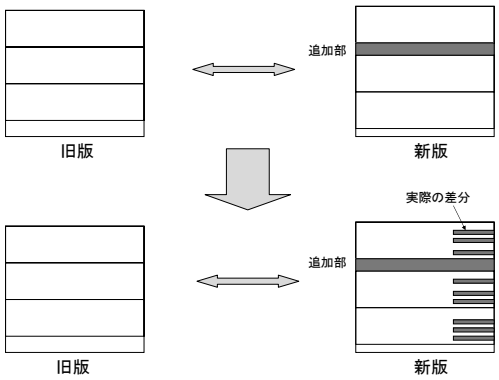

図 2: 修正による全体への影響

挿入, 削除したところ以外にも影響が及ぶ．絶対アドレスで指定している部分は, そのアドレス情報にずれが生じる．相対アドレスでアクセスするような場合でも相対アドレス指定している間にずれが生じているとやはり影響を受ける．

この影響は，必ずアドレスを表現する部分に現れる．しかもずれるといっても，出荷後のソフトウェアでは，大きなずれは考えにくく，アドレス指定の下位数ビットにずれを生じる可能性が高い．この特徴は,CPU のアーキテクチャによっては修正点が命令長の一部に集まるということを示している．

そこで, 実際に出荷済みの携帯電話のソフトウェアに, わずかな修正を入れてみた場合と, 比較的大きな修正を入れてみた場合とでその傾向を調べた．

表 1: 差分の出現位置（修正がわずかな場合)

バイト位置	1	2	3	4
差異サイズ	490	490	558	2243

バージョン２→バージョン１

バイト位置	1	2	3	4
差異サイズ	516	516	580	2240

表 2: 差分の出現位置（大きな修正の場合）

バージョン１→バージョン３

バイト位置	1	2	3	4
差異サイズ	119515	120056	128889	282219

バージョン３→バージョン１

バイト位置	1	2	3	4
差異サイズ	121752	122274	131775	282800

その結果を表 1,2に示す．ソフトウエェアの規模としては全体で 16M バイト程度である． 表 1は修正がわずかな場合であり, バージョン１から２の場合とその逆を調べた結果である． 表 2は修正が大きかった場合であり, バージョン１から３の場合とその逆を調べた結果である．それぞれワード単位で区切って, ワードの中の何バイト目に差分が現れたかを示している．いずれの場合も 4 バイト目に差分が大きく出ていることがわかる．M32R という CPU のアーキテクチャ上, アドレスを示すデータの下位の位置が 4 バイト目に現れるからである．

さらに, それぞれの場合において, 連続 4 バイト即ち, ワードごと異なる場合は何ワードあるかを調べた．その結果を表 3に示す．これらの結果から, 連続して差異が発生している部位, 即ち修正そのもの

表 3: 連続した差分バイト数

対象バージョン	バイト数
$1 \rightarrow 2$	472
$2 \rightarrow 1$	464
$1 \rightarrow 3$	117316
$3 \rightarrow 1$	119418

ののデータとその修正によるアドレスずれの影響を受けた部分が明確に区別できることがわかる.

3.1 差分表現形式

旧版から新版を作するための差分情報は, 一般に以下に示す２つの情報で表現できる.

1. COPY 開始アドレス, 長さ
   どこから, どの長さコピーするかを示す.

2. DATA データ長と続くデータ列
   新たに加えるべきデータを示す.

携帯電話のフラッシュ ROM はコストの関係上二重化するわけではなく, また RAM もフラッシュ ROM に比べて小さい. そのため, 旧版から新版に書き換えて行く過程で, すでに書き換えてしまったデータからコピーすることがないような制約が必要になる. このような制約条件を満足するように差分を表現する.

ここで,COPY コマンドの数を n とし, データコマンドの数を m とし, データコマンドに続くデータ列の平均の長さを l とする. 各コマンドは 1 バイトで表現でき, データコマンドは gdiff のようにデータ長そのものがコマンドを意味することができ, アドレスは 4 バイトで表現でき, 長さは 1 バイトで表現できるとすると, 差分サイズ $diffsize$ は以下の式で表すことができる.

$$diffsize = n(4+1+1) + m*(l+1)$$
$$diffsize = 6n + m(l+1)$$

ここで, 新版と旧版で異なる部分は, 合計で $m*l$ であるから, コピーコマンドの数を小さくすればするほど差分サイズは小さくなることを示している. その例を図 3, 図 4に示す.

図 3は差が中ほどに一部ある場合で, この場合は,COPY コマンドが２つにる. 逆に図 4では差が最後にあるため,COPY コマンドは一つで済む. このように, 同じ差があるとしてもその位置により COPY のコマンドの数が変わり, 結果的として差分のサイズが変わる.

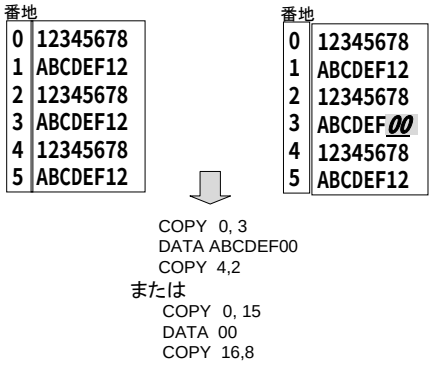

図 3: 修正の影響

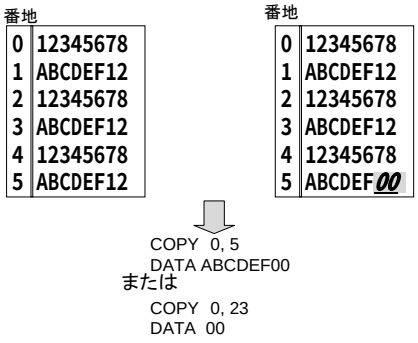

図 4: 修正の影響を抑えた例

4 改良差分抽出方式

差分を表現する上では COPY コマンドの数を減らすことがポイントである. そのための手段として以下の２つの方法を提案する.

1. アドレス空間の変換
   前節で着目した差分の出る位置に着目し, アドレス空間の変換により, 均一な差分の分布から, 不均一な分布に変換することにより COPY コマンドを減らす方法.

2. コピーのマクロ的変換
   コピーと修正をシーケンシャルに実行するのではなく, マクロな視点からずれを表現した後に, 修正点のみを集めて送る方法.

4.1 論理空間による差分抽出方式

アドレス空間の変換を次式を元に行う.

$$new = \lceil old/4 \rceil + \lfloor n/4 \rfloor * mod(old/4)$$

ここで, 各変数の意味を以下に示す.

new : アドレス変換後のアドレス
old : オリジナルの物理的アドレス
n: 全データサイズ

例えば, 全データが 100 バイトだとすると,0 番地は 0 番地,1 番地は 25 番地,5 番地は 1 番地となる. 表 1, 2の例で, アドレス変換した後の差分の出現バイト位置をそれぞれ表 4 , 5に示す.

また, その場合の連続した差分のワード数に関して, 表 6に示す. この結果から, 修正がわずかな場合で更新のための修正による位置ずれの影響が多きい場合も, 修正が多岐に渡り, 更新のための修正そのものの影響の方が大きい場合も差分の出現位置のバイト位置による偏りが少なくなることがわかる.

また, 修正がわずかな場合は 4 バイト連続して差異が現れる場合が多くなり, 差異の部分が集中していることがわかる. しかし, 修正量が多い場合は,4 バイト連続した差異の部分は逆に減り, 期待した効果はあまり得られないと考えられる.

4.2 マクロ的コピーによる差分表現方式

さらに, マクロ的視点のコピーを見つけることにより, より差分を小さく表現する方法を検討した. 図 5に示すようにマクロ的コピーを利用することにより,COPY コマンドを減らすことができる. ただし,COPY コマンドは減るかわりに, データコマンドをアドレススキップしながら発行する必要がある. 差分サイズは COPY コマンド数 n, データコマンド m, 平均長 l で

$$diffsize = 6n + m * (l + 1)$$

と表現できた.

ここで, マクロ的な COPY コマンド数を x とし, マクロコピー中のスキップの平均数を a とすると, コピーコマンドの差分サイズは

$$copy = 6\mathrm{x}$$

となる. ここで, スキップの表現には, 長さの情報だけがあれば良い. 多くの場合,1 バイトもあれば表現できるため, スキップ表現の大きさは,$a(1+1)$ である. 即ち, 差分サイズは,

$$diffsize = 6x + 2ax + m * (l + 1)$$

となる. よって,$6n$ と $(6 + 2a)x$ の差が差分サイズの大きさの違いとなる.

$$y = 6n/(6 + 2a)x$$

$n = ax$ であるので以下となる.

$$y = 6ax/(6 + 2a)x$$
$$y = 1/a + 1/3$$

表 4: アドレス変換後差分の出現位置（わずかな修正の場合）

バージョン１→バージョン２

バイト位置	1	2	3	4
差異サイズ	1442	1435	1406	1433

バージョン２→バージョン１

バイト位置	1	2	3	4
差異サイズ	1456	1447	1467	1430

表 5: アドレス変換後差分の出現位置（大きな修正の場合）

バージョン１→バージョン３

バイト位置	1	2	3	4
差異サイズ	143832	143089	143163	141364

バージョン３→バージョン１

バイト位置	1	2	3	4
差異サイズ	141298	143567	143577	144329

表 6: アドレス変換後連続した差分バイトの数

対象バージョン	バイト数
1 → 2	1012
2 → 1	1012
1 → 3	101748
3 → 1	101748

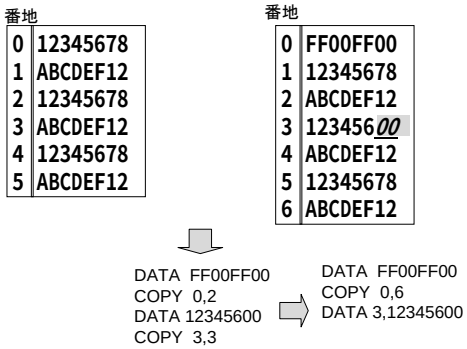

図 5: マクロ的コピー利用による効果

従って y は単調減少となり,a が大きいほど差分サイズは小さくなることがわかる．ただし,$a > 3/2$ 以上でなければ逆に差分サイズは大きくなるが、多くの場合は効果があることを示している．

5 評価

5.1 アドレス変換による効果

提案した２つの方法では, 修正の大, 少によって差異の分布がどうなるかを既に示した．そこで, それぞれに対する差分のサイズ評価を行う． 表 7に差分サイズ一覧を示す．修正部が小さい場合で, 約 30% 差分サイズを削減できている． 修正部が大きい場合は, 約 40% 差分サイズを削減できている．

修正部が大きい場合は, 連続４バイトの差異が減っていたにもかかわらず差分データサイズという観点では大きな効果が出ている．この理由は, 連続 4 バイトだけに着目していたためで, COPY コマンドの数が減っている即ちデータコマンドの数も減っているかどうかをそのままでは示していないためだ

表 7: 差分サイズ

対象 バージョン	アドレス変換前 差分サイズ	アドレス変換後 差分サイズ
1 → 2	17434	12248
2 → 1	17434	11697
1 → 3	1917130	1130463
3 → 1	1919958	1129068

表 8: データコマンド数

対象 バージョン	アドレス変換前 差分サイズ	アドレス変換後 差分サイズ
1 → 2	1884	790
2 → 1	1891	791
1 → 3	172380	78383
3 → 1	171585	79124

と考えられる．

表 8に, それぞれの場合のデータコマンド数を示す．差異部分が変わるわけではないため, データコマンド数が減ることは, データコマンドで送られる各データ長の平均長が大きくなっていることを示しており, 全体の差分サイズの大きさを小さくしていることがわかる．

5.2 マクロ変換による効果

さらにマクロ変換を掛けてみた． その結果を表 9に示す． アドレス変換をかけたあとに更にマクロ変換をかけた場合に, 修正が少ない場合で 20% の差分サイズ削減, 修正が多い場合で 24% の削減ができている．アドレス変換およびマクロ変換をか

表 9: マクロ変換後差分サイズ

対象 バージョン	アドレス変換後 差分 (バイト)	マクロ変換後 差分 (バイト)
1 → 2	12248	9896
2 → 1	11697	9351
1 → 3	1130463	866291
3 → 1	1129068	862632

表 10: 実データでの差異位置

バイト位置	1	2	3	4
差異サイズ	7472	7491	8480	28001

表 11: 実データでの差分サイズ

	アドレス変換前	アドレス変換後
データコマンド数	21532	7315
差分サイズ	202792 バイト	132835 バイト

ける前から比べると差分サイズは半分以下に抑えられることがわかる.

5.3 出荷直前ソフトウェアによる評価

本提案を出荷前の状態のソフトウェアの版に対して, 実際の不具合の修正を行ったもので評価を試みた. 出荷後にはこれほどの不具合は出ないであろうという段階のものである. 表 10, 11に結果を示す. この結果からもアドレス変換, マクロ変換の有効性が確認できる.

6 考察

6.1 アドレス変換に関して

差分データサイズに関しては一定の効果が得られることを確認した. しかしながら, 実データでは, 効果がテストデータに比べて上がっていない. この原因は, 以下のことが考えられる.

1. M32R の CPU アーキテクチャは 4 バイト命令と 2 バイト命令が混在しており, 必ずしも一定のバイト位置にアドレス部が出るとは限らない. しかし, ジャンプ先アドレスはワードバウンダリに合わなければならないという制約条件から多くの場合は,NOP 命令などが入ることになり, 実質的には,4 バイトの固定命令と扱っても差し支えないはずであるが、若干この部分が実データでは影響があったと考えられる.
2. 試験データでは偏った修正を施している部分があるが, 実データでは修正が広範囲に分布されていると想定される. そのため, 影響の出方が変わったと考えられる.

上記 2 点を考慮し, さらに, 命令コードを解析した上でアドレス部を特定して変換する方法, 連続して差異がでる部分を見つけ, この部分までで, 一旦アドレス変換を掛ける方法が考えられる. この方法によれば, 連続して差異があった部分が変換をかけたことによって, 逆に分散するということを防ぐことができ, 差分を小さくすることができるはずである.

また, アドレス変換を行う上では, フラッシュ ROM の構造上との関連で注意するべき点もある. フラッシュ ROM はイレースブロックという単位でまとめて消去し書き込みをする必要がある. そのため, 論理アドレス上での処理を物理アドレスに戻す段階で破綻しないような仕組みが必要である. 即ち, メモリの制約条件を厳しく考える必要がある.

RAM 部がフラッシュ ROM より極端に少ない携帯電話では, 全体をまとめて変換することはできず, 一部ずつ変換するという処理になると想定される. このような処理を入れた場合の効果の考察に関しては今後の課題である.

6.2 コピーのマクロ変換に関して

コピー命令をまとめてマクロ的に変換する方法は有効であった. しかし, 大きな視点でずれを発見するのは差分抽出する上で難しい. ここでは, 一旦差分を抽出した上で, コピーのずれ幅を見て同じ幅のものをまとめてマクロコピーという方法をとっている. しかしながら, この手法では, 一旦できた差分を再度全部読み直し変換する必要がある上に, マクロ的なコピーの中に小規模な別のコピーが発生しているとうまく抽出できないという問題もある.

コード生成過程の情報との連携により, 効果的なマクロコピーを発見できるのであれば, 一定の計算時間の中でさらに差分を小さくすることが可能ではないかと考える.

また, マクロ的なコピーをしてから, 残りの部分の差分を適用するという方法では,RAM が十分にない場合に, フラッシュ ROM に二重に書き込みが発生する可能性がある. これは更新時間が長くなる

ことを意味しており好ましくない．この点は今後の課題と考えている．

7　まとめ

本論文では携帯電話の S/W 構造,CPU のアーキテクチャに着目して,S/W のバージョン間の差分を効果的に小さく表現する手法に関して提案した．アドレスを物理アドレスから論理アドレスに変換して扱うだけで効果が得られることを示した．また, 差分表現上のコピーコマンドをマクロ的な視点で扱うだけでも差分量の削減に効果があることを示した．

今後, さらなる差分を小さくするための工夫の他, アドレス変換をする上で問題となる集中した差分を逆に分散してしまう点や, メモリ上の制約問題, およびマクロ変換における抽出時間の問題と二重書き込みの問題に関して整理して行く予定である．

参考文献

[1] Masato Takeichi et al.,"Bug Fix of Mobile Terminal Software using Download OTA", The Asian-Pacific Network Operations and Management Symposium 2003.

[2] 栗原まり子他, "携帯電話 SW のバージョン間差分データに関する検討", 情報処理学会, DICOMO'03, 074 (2003).

[3] 清原他,"携帯電話の SW 更新を目的としたモジュール分割に関する検討", 情報処理学会, DICOMO'03, 073(2003)

[4] James W. Hunt and M.D.Mcillroy,"An algorithm for differential file comparison", Tech. Rep. 41,AT & T Bell Laboratories,Inc.,Murray Hill,NJ. 1976

[5] James W. Hunt and Thomas G. Szymanski,"A Fast Algorithm for Computing Longuest Common Subsequences",Communications of the ACM, Vol.20,No.5,pp.351-353,1977

[6] W.F.Tichy,"The string-to-string correction problem with block moves",ACM Transactions on Computer systems, Vol2,No.4,PP.309-321,1984

[7] W. Obst,"Delta technique and string-to-string correction",Proceedings of the 1st ESEC of the ACM, Vol.20,No.5,pp.351-353,1977

[8] James J. Hunt,"Delta Algorithms: An empirical Analysis",ACM Transactions on Software Engineering and Methodlogy, Vol.7,No.2,PP. 192-214,1998

[9] Andrew Tridgell et al.,"The rsync algorithm",ANU,TR-CS-96-05

[10] http://www.w3.org/TR/NOTE-gdiff-19970901

[11] David G. Korn and Kiem-phonfg Vo,"Engineering a Diffrencing and Compression Data Format", Proceedings of the 2002 USENIX Annual Technical Conference,2002

[12] Kohei Terazono et al., "An Extended Delta Compression Algorithm and the Recovery of Failed Updating in Embedded Systems",DCC2004,

[13] Randal C. Burns and Darrell D.E. Long,"In-Place Reconstruction of Version Differences", IEEE Transactions on Knowledge and Data Engineering,Vol.15,No.4,PP.973-984,2003

[14] 高田他,"4M バイト DRAM 内蔵 32 ビット RISC マイクロコントローラ"M32Rx/D", 三菱電機技報,Vol.73, No.3 PP.182-185,1999