Vol.2013-HPC-140 No.45
2013/8/2

メモリトレース解析によるアクセスパターンのモデル化

松原 裕貴[1,2,a)]　佐藤 幸紀[1,2,b)]

概要：プロセッサの速度向上に対するメモリ速度の向上が追いついていないといったメモリウォール問題は、近年ではマルチコア化に伴いますます解決すべき重要な問題として認識されている。この問題を解決するための手法として、キャッシュミスの改善やメモリ階層に着目したチューニングは有効であると考えられるが、大規模な処理におけるメモリアクセスは大量かつ複雑であり、その解析は自動化が求められている。本論文では、ループ階層構造の検出とメモリアクセストレースの取得を可能とする実駆動型アプリケーション解析ツールを用いてメモリアクセス命令毎のデータアクセスパターンの動的なモデル化手法の提案を行う。本手法により、プログラマに対してアクセスの傾向のモデルを抽象的に提示することが可能となり、性能解析の効率を上げることが期待される。評価実験として、姫野ベンチマークのメモリトレースを対象としたモデル化を行い、その有用性の確認を行った。

1. はじめに

高性能計算機分野で用いられるアプリケーションの計算量は非常に膨大であるが、同時に多くのメモリアクセスも生じる。近年では、CPU の性能向上に対するメモリの性能向上が追いつかないといったメモリウォール問題 [1] も指摘されており、メモリアクセスによって生じるレイテンシ等の改善はアプリケーションの実行時間を短縮する上で重要な課題である。更に高性能計算機のアーキテクチャによってはメモリの階層構造が深くなるため、メモリアクセスの改善による効果は大きと考えられる。メモリアクセス効率を考慮したアプリケーション開発においてメモリアクセスのトレース解析が必要となるが、HPC アプリケーションにおいてメモリトレースの量は膨大かつ複雑であるといった課題が存在する。このため、実際の処理においてどのようなメモリアクセスが生じていたかアプリケーションの開発者が直感的に理解し分析できるよう、アクセスパターンの発見と簡素化により抽象化を図る必要があると考えられる。

本論文では、実駆動型アプリケーション解析ツール Exana (Execution-driven application analyisis tool) [2] から得られるメモリアクセストレースを解析対象としたメモリアクセス解析ツールの提案を行う。メモリアクセス解析ツールでは、メモリアクセス命令毎にアクセスされたデータアドレスの解析を行い一定のルールに従ったアクセスをアクセスパターンとして検知し、アクセスパターンのモデル化を行う。アプリケーション開発において、規則性を持つデータアクセスを行うことは珍しくなく、一定のルールを検知し、そのパターンをモデル化することで解析に必要なデータ量の削減と抽象度の向上によるアクセス解析の効率化が期待できる。また、開発したメモリアクセス解析ツールでは、アプリケーションの実行途中でもどのメモリアクセス命令においてどのようなアクセスパターンが生じているか解析できるようオンライン処理によるアクセスパターンのモデル化を実現している。これにより、実駆動型アプリケーション解析ツールとの連携も実現可能となる。更にアクセスパターンの分類分けを行い、モデル化されたパターンの分類の自動判別も行っている。

ここで、本論文の構成を述べる。メモリアクセスの分類とアクセスパターンのモデル化のルールについて述べ、オンライン処理によるアクセスパターンのモデル化手法の説明を簡素化したコードと例を用いて説明する。最後に、姫野ベンチマークより実駆動型アプリケーション解析ツールで取得したメモリトレースのモデル化の例を示し、データ量の比較から有用性の検証を行う。

2. メモリアクセスパターンのモデル化と分類

実駆動型アプリケーション解析ツールから得られるメモリトレースにはメモリアクセス命令毎に Read/Write 情報、アクセスしたデータサイズ、アクセスしたデータのアドレスがアクセスが生じた順で記述されており、各々のメモリアクセス命令がデータに対してどのようなアクセスを

[1] 北陸先端科学技術大学院大学 情報社会基盤研究センター
[2] JST CREST
[a)] yuuki-mt@jaist.ac.jp
[b)] yukinori@jaist.ac.jp

行ったかがわかる。このトレース情報を用いてアクセスパターンの検知とモデル化を行う。また、データアクセスは三つの種類に分類できる。一つ目は固定アドレスに対するアクセス、二つ目は連続したデータアドレスに対するアクセス、三つ目はアクセスされたデータ間にオフセットが存在するアクセスである。

本論文ではこれらのアクセスを固定アクセス、シーケンシャルアクセス、ストライドアクセスとしてアクセスパターンの分類分けを行う。例として、実駆動型アプリケーション解析ツールから得られるメモリトレースとアクセスの分類を示す (図 1)。トレース情報は左から順に Read/Write 情報、アクセスしたデータサイズ、メモリアクセス命令のアドレス、アクセスしたデータアドレスとなっており、loopID から始まる行はループ処理関係の情報となっているが現段階で loopID の情報はメモリアクセス解析には利用していない。まず、0x40054b 番地のメモリ操作命令は 0x7fffffffd054 番地のデータアドレスにしかアクセスしていないため、固定アクセス (Fix) に分類される。0x400527 番地のメモリ操作命令は 4byte 毎の連続したデータアドレスへのアクセスであるため、シーケンシャルアクセス (Sequential) に分類される。0x400533 番地のメモリ操作命令はデータアクセス間に 4byte のオフセットが存在するため、ストライドアクセス (Stride) に分類される。また、0x400533 番地のメモリ操作命令はシーケンシャルアクセスも含む。

ここで、アクセスパターンのモデル化について述べる。あるメモリ操作命令の固定アクセスまたはシーケンシャルアクセスのアクセスサイズをアクセスパターンの先頭データのサイズ D としたとき、次にアクセスされるデータサイズを $D = D_n$ とし、データ間のオフセットを O_n としたとき、D 以降の O_c と D_c のセットの連続回数を $c(0 \leq c)$ として表わされるデータアクセスをアクセスパターンとする。また、O_c と D_c のセットが出現する範囲を連続区間とする。なお、$c = 0$ の場合はデータ間のオフセットとデータサイズは存在しないため、先頭のデータサイズのみと判断でき、固定アクセスかシーケンシャルアクセスに分類される。一方、$c > 0$ の場合はストライドアクセスに分類される。更に、先頭データのアドレスとアクセスパターンが同じデータアクセスが連続的に出現した回数をアクセスパターンのリピート回数として $r(r > 1)$ で表す。これらの表現を用いたアクセスパターンモデルのフォーマットは $[[D] < \_O_c\_[D_c] > (c)](r)$ となる。最も外側の"[ ]"はアクセスパターンを示し、(r) はアクセスパターンのリピート回数を示す。内側の"[ ]"はアクセスされたデータサイズを表し、"_ _"はオフセットを表しており、"< >"はデータ間オフセットとデータサイズのセットを表し、(c) はセットの連続回数を示している。規則性のあるオフセットとデータアクセスをパターンの検知ルールとすることで連続回数やリピート回数による表現が可能となりデータ量の削減と抽象化が実現できる。

図 1 のメモリトレースをモデル化した結果を図 2 に示す。モデル化はメモリ操作命令毎に行われる。モデル化された結果の説明を行う。左のオフセットはアクセスパターン間のオフセットを表し、始めのアクセスパターンではアクセスパターン間のオフセットは 0 となる。次にアクセスパターンの分類とアクセスパターンの先頭データのアドレスを示し、その後にモデル化されたアクセスパターンを示す。なお、固定アクセスとシーケンシャルアクセスの場合の"( )"はリピート回数を表す。0x400533 番地のメモリ操作命令には二つのアクセスパターンが含まれており、これらのアクセスパターン間のオフセットはアクセスパターン間オフセットとして"_12_"として表されている。

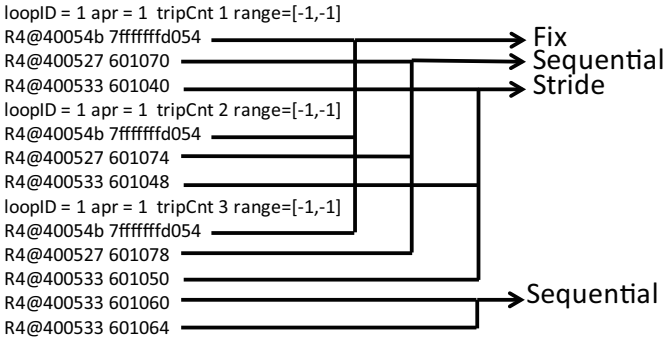

図 1 メモリトレースとアクセスの分類例

```
R4@40054b = {
        _0_Fix:7fffffffd054 [4](3)
}

R4@400527 = {
        _0_Sequential:601070 [12](1)
}

R4@400533 = {
        _0_Stride:601040 [[4]<_4_[4]>(2)](1)
        _12_Sequential:60160 [8](1)
}
```

図 2 メモリトレースのモデル化の例

3. アクセスパターンのモデル化アルゴリズム

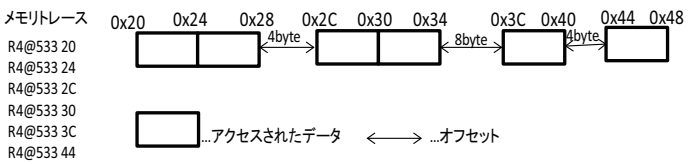

図 3 メモリトレースの例

本節では、メモリトレースからアクセスパターンをモデル化する手法を例 (図 3) を用いて説明する。また、それぞれの処理はメモリ操作命令毎にトレースデータが読み込まれる度に行われる。

3.1 データ構造の初期化

モデル化の処理を行うために、初めて読み込まれたメモリ操作命令を検知した際に 3 つのデータ構造を用意する。一つは、オフセット (of) とアクセスされたデータの先頭アドレス (sa) とシーケンシャルアクセスされた場合にアクセスされるデータの先頭アドレス (ncda) で構成される処理中のデータアクセスに関する情報を格納する一時データ temp となる。初期化の処理では、of は 0 とし、読み込まれたトレースデータから sa と ncda を入力する。ncda は読み込まれたデータに一回のデータアクセスサイズが加算されたものとなる。二つ目は、アクセスパターン間オフセット (aof) とアクセスパターンの開始データの先頭アドレス (apsa)、先頭データ及び連続区間内のデータサイズ (ds)、データ間オフセット (dof)、連続回数 (cc)、リピート回数 (rc) により構築されたモデル化の候補データ candidate、三つ目は、モデル化の候補データが確定した際に保存される確定データ deside である。初期化の処理では、candidate と deside のデータは全て 0 となる。

例 (図 3) において始めに読み込まれるトレースデータは R4@533 20 であり、初期化されたデータは temp[0,0x20,0x24],candidate[0,0,0,0,0,0],deside[[0,0,0,0,0,0]] となる。

3.2 モデル化処理

3 つのデータ構造 temp,candidate,deside を用いて、読み込まれたトレース情報をモデル化する処理の説明を行う。また、モデル化処理の概要を示す関数を図 4 に示す。図 4 の仮引数 cda は読み込まれたトレースのデータアクセスの先頭アドレスを示し、ncda は cda に一回のアクセスサイズを加算したアドレスである。

3.2.1 シーケンシャルアクセス検知時の処理

図 4 のモデル化関数 2 行目では読み込んだトレースのデータアクセスがシーケンシャルアクセスであるか判定処理を行いう。読み込まれたトレースのデータアクセスの先頭アドレス cda が前の処理で得た temp の ncda のアドレスと一致するか確認する。ncda はトレースが読み込まれた際に、メモリ操作命令の一回のアクセスサイズをデータアドレスに加算したものであり、加算されたアドレスは次のアクセスがシーケンシャルなものであった場合のデータの先頭アドレスとなるため、nca と ncda が一致した場合はシーケンシャルアクセスが生じたことがわかる。次のアクセスもシーケンシャルアクセスである可能性があるため、3 行目で temp の ncda の更新を行う。

```
1. def make_access_model(cda,ncda):
2.     if cda == temp[2]:
3.         temp[2] = ncda
4.     else:
5.         offset = calc_OffsetSize( temp[2],cda)
6.         ads = calc_DataSize(temp[1],temp[2])
7.         if candidate[3] == temp[0] and candidate[2] == ads:
8.             if det_cur(candidate[1],temp[1]):
9.                 candidate[5] +=1
10.            else:
11.                candidate[4] +=1
12.        else:
13.            if det_loop(deside[-1],candidate[2]):
14.                deside[-1][5] +=1
15.            else:
16.                deside.append(candidate)
17.            reset_candidate (candidate[2],temp[0],temp[1],offset)
18.        reset_temp(temp,offset,cda,ncda)
```

図 4 モデル化の処理関数

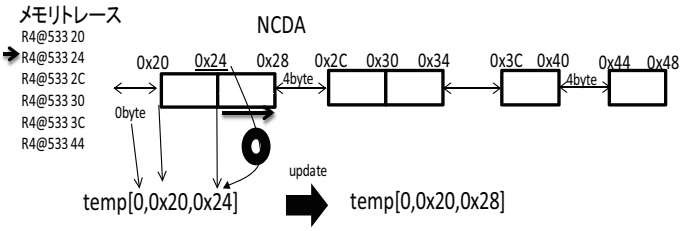

図 5 シーケンシャルアクセス時の処理例

例 (図 5) では、R4@533 24 のトレース処理に移行し、シーケンシャルアクセスの検知を行う。前回取得した ncda と読み込まれたデータアドレスが一致するため、temp の NCDA をデータアドレスに 4byte 足した 0x28 で更新する。

3.2.2 ストライドアクセス検知時の処理

図 4 のモデル化関数 4 行目以降はストライドアクセスの処理を行う。シーケンシャルアクセスではないため、アクセスされたデータ間にオフセットが存在する。このとき、図 4 5 行目で読み込んだメモリ操作命令のデータアドレス cda から、temp の ncda を引くことによりデータ間のオフセットがわかる。さらに、図 4 6 行目では temp の ncda から temp に記録されたデータの先頭アドレス sa を引くことによってアクセスされたデータサイズも取得する。

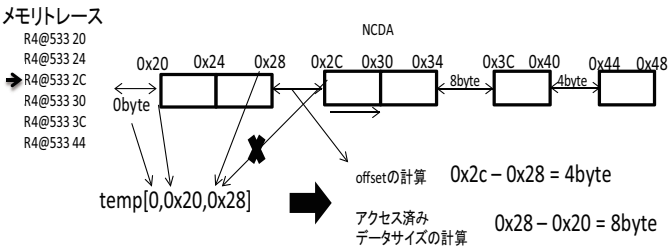

図 6 ストライド時の処理例:オフセットとデータサイズの計算

例 (図 6) では、ストライドが検知されたため、データサイズとオフセットの計算を行い、データサイズ 8byte とオフセット 4byte が算出される。図 4 7 行目では temp のオフセットと算出したアクセス済みデータサイズと candidate

のデータ間オフセットとデータサイズが一致するか調べる。一致する場合はモデル化の候補パターンの連続区間が検知されたことになり、連続回数をカウントする (図 4 11 行目)。ただし、候補パターンがストライドではなく、固定アクセスまたはシーケンシャルアクセスに分類される場合、リピートアクセスの可能性があるため、図 4 8 行目で、候補パターンと temp のデータ開始アドレスが一致するか調べて一致する場合は図 4 9 行目でリピートアクセスのカウントを行う。例 (図 7) では、候補パターン candidate のデー

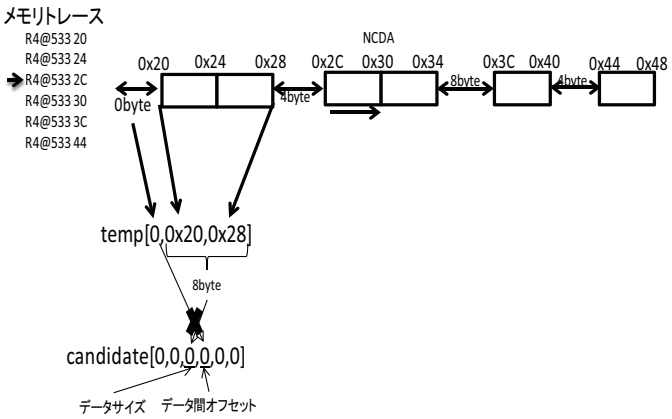

図 7 ストライド時の処理例:連続区間の検知

タ間オフセットとデータサイズは共に 0byte で、temp のオフセットは 0byte であるが前の処理で得た計算したデータサイズは 8byte なので連続区間の条件に一致しない。このため、候補パターンを確定データに追加する処理に移行する (図 4 12 行目)。候補パターンの確定データへの追加処理では候補パターンが確定データ内の直前のアクセスパターンとリピート関係であるか調べる (図 4 13 行目)。リピート関係の判定処理は、確定済みのアクセスパターンの候補パターンの開始データの先頭アドレスとデータサイズが一致しており、更に連続区間がない場合とデータ間オフセットと連続区間の回数が同じ場合にリピート関係が存在すると判定される。リピート関係が存在する場合は、確定データ内のアクセスパターンのリピート回数をカウントし (図 4 14 行目)、リピート関係ではない場合は候補パターンを新たにアクセスパターンとして追加する (図 4 16 行目)。

例 (図 8) では、候補パターンの先頭データのアドレスとデータサイズが一致し、更に連続区間の繰り返し回数も 0 であるため、候補パターン [0,0,0,0,0,0] は新たに確定データへは追加されず、確定データのアクセスパターンのリピート回数にカウンされる。なお、初期化によって設定された候補パターンと確定データのアクセスパターンは以降の処理で、新たに候補パターンを準備するために利用されるデータであり、出力結果には確定データの最初のアクセスパターン [0,0,0,0,0,1] は出力されない。

候補パターンの確定データへの追加処理が終了した際に

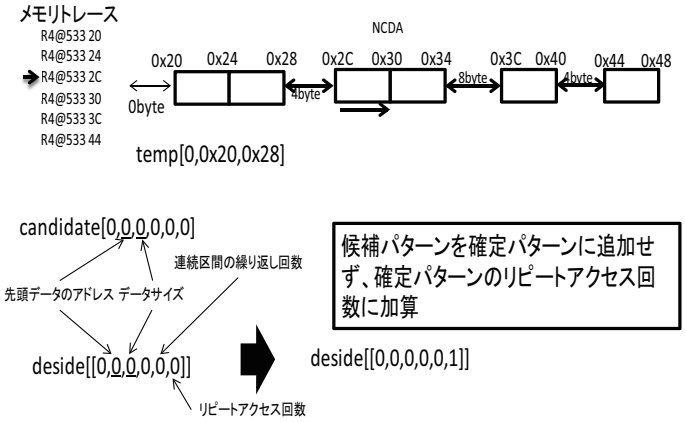

図 8 ストライド時の処理例:リピート関係の判定

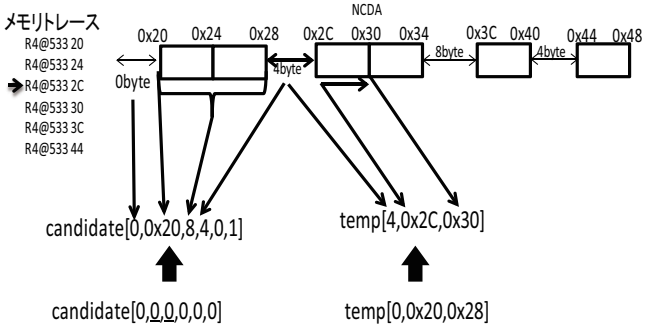

図 9 ストライド時の処理例:候補パターンと一時データの更新

temp のオフセットとデータの先頭アドレスと前の処理で計算したオフセットにより次の処理が終了した際に候補パターンの更新を行う (図 4 17 行目)。また、ストライド検知時の処理が終了した際にトレースデータのデータの直前のオフセットと仮引数のトレースデータのデータの先頭アドレス cda と仮引数の次にシーケンシャルアクセスされる可能性のあるアドレス ncda により、一時データ temp を更新する (図 4 18 行目)。例 (図 9) では候補パターンは [0,0x20,8,4,0,1] と更新され、一時データは [4,0x2C,0x30] と更新される。

以降は図 3 の例を対象として説明を行う。R4@533 30 のトレースデータはシーケンシャルアクセスと判断されるため、temp の NCDA を更新して [4,0x2C,0x34] となる。続けて、R4@533 3C を読み込んだ際に先頭のデータアドレス 0x3C と temp の NCDA 0x34 が一致しないため、ストライドを検知する。

例 (図 10) では、一時データを計算して得たデータサイズ 8byte とオフセット 4byte が候補パターンのデータサイズ 8byte とデータ間オフセットと一致し、更に先頭データのアドレスが同じではないため、連続区間と判断されて候補パターンの連続アクセス回数にカウントされる。ストライド処理終了後、一時データを [8,0x3C,0x40] に更新する。最後に、R4@533 44 のトレースデータを読み込み、ストライドが検知される。

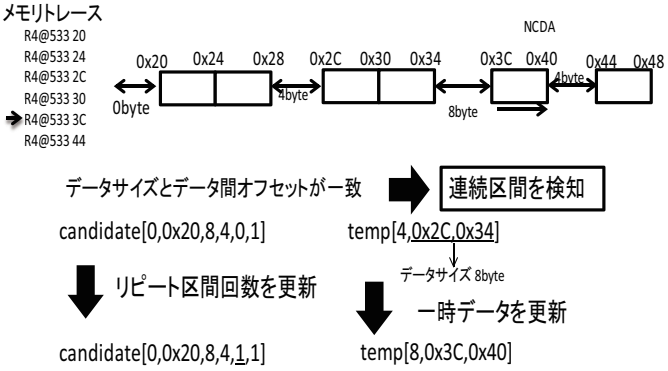

図 10 ストライド時の処理例:連続区間の検知

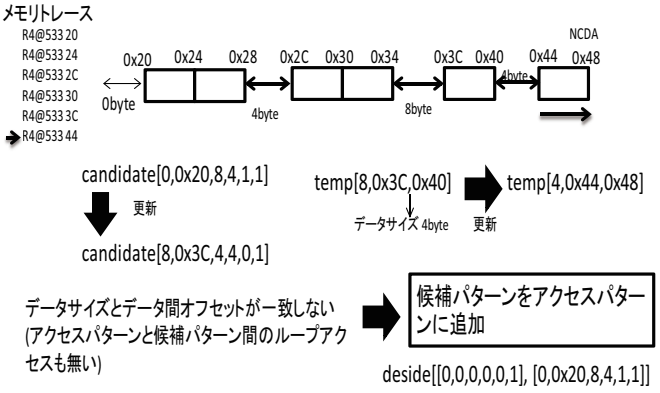

図 11 ストライド時の処理例:アクセスパターンの追加

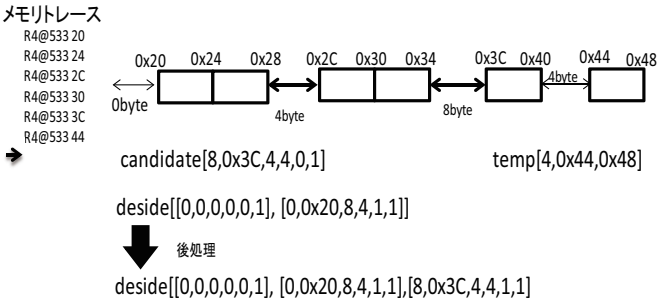

図 12 後処理の例

例 (図 11) では、候補パターンで連続区間は検知されず、アクセスパターンと候補パターン間でもリピート関係がないため、決定データに候補パターンが追加される。また、候補パターンと一時データの更新を行う。R4@533 に関しては全てのトレースデータを読み込み終わった状態となるため、モデル化処理を終了する。

3.3 後処理

モデル化処理終了後、各メモリ操作命令の一時データと候補パターンがアクセスパターンに追加されずに残る。これはストライドを検知しなければ、アクセスパターンの追加処理が行われないためであり、最後に同様のモデル化処理を残りのデータに対して行う。

例 (図 12) では、候補パターンとして [8,0x3C,4,4,0,1] と一時データとして [4,0x44,0x48] が残っており、モデル化処理と同様の後処理を行うことで、モデル化されたアクセスパターンとして deside[[0,0,0,0,0,1], [0,0x20,8,4,1,1],[8,0x3C,4,4,1,1]] が得られる。

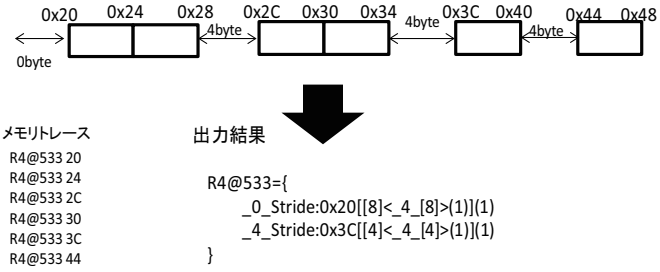

図 13 モデル化の最終出力結果の例

図 13 に図 3 をモデル化した最終結果と示す。なお、決定データ内の最初のアクセスパターンは他のデータを準備するために利用されるデータであるため、最終結果には含まない。

4. 評価実験

評価実験として、実駆動型アプリケーション解析ツールにより得た姫野ベンチマークの特定のループ内のメモリトレース loop11 と loop10 の二つを本論文で提案したメモリアクセス解析ツールによりモデル化を行い、トレースデータ数に対するモデル化後のデータ数を示し、データ数の圧縮率を調べる。また、モデル化したデータはアクセスパターン間オフセットとアクセスパターンのモデルをセットとしてカウントする。

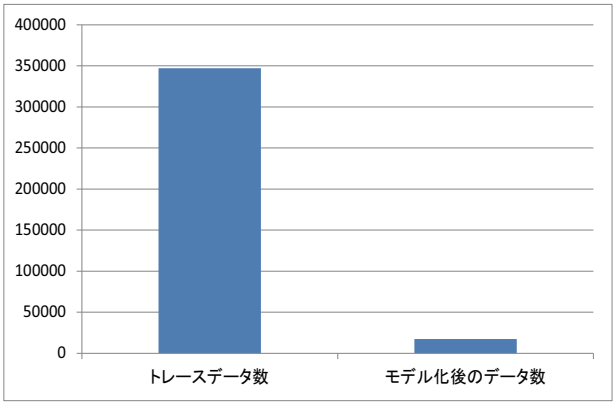

図 14 メモリトレース loop10 のトレースデータ数とモデル化後のデータ数

loop10 と loop11 のトレースファイルの結果を図 14, 図 15 に示す。結果として loop10 のトレースデータ数が 347094 個に対し、モデル化後のデータ数は 17404 個となり、データ数の圧縮率は約 95%となった。また、loop11 のトレースデータ数が 2734 個に対し、モデル化後のデータ数は 146 個となり、データ数の圧縮率は約 95%となった。この結果から、本論文で提案したメモリアクセス解析ツールは高

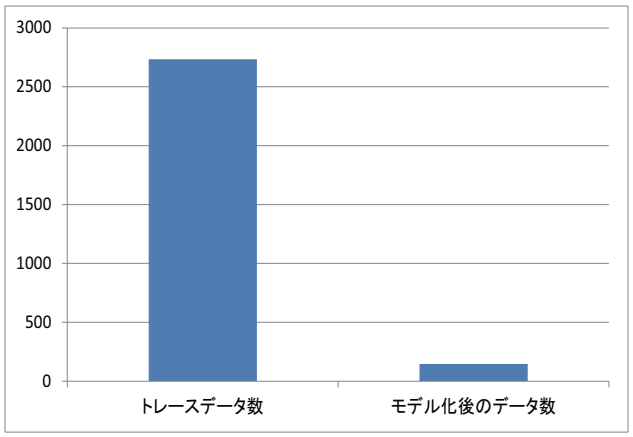

図 15 メモリトレース loop11 のトレースデータ数とモデル化後のデータ数

いデータ数の圧縮率を実現可能であることがわかった。また、この削減率から loop10 と loop11 のデータアクセスはほとんどで規則性がある可能性が高いと考察できる。

メモリアクセス解析ツールでは複数のモデル化データを持つメモリ操作命令のカウントも行っており、loop11 では 0 となったが loop10 では全てのメモリ操作命令で複数のモデル化データがあるといった結果が得られた。このため、loop10 のモデル化データの調査を行った結果、複雑なアクセスパターンが存在することがわかった。

loop10 で発見された特徴的なアクセスパターンの一例を図 16 に示す。図 16 では線で区切られたモデル化された

```
R4@40211e={
        _0_Fix:0x476e6c0[4](1)
        _1020_Sequential:0x476eac0[8](1)
        _1016_Sequential:0x476eec0[12](1)
        _1012_Sequential:0x476f2c0[16](1)
        _1008_Sequential:0x476f6c0[20](1)
        _1004_Sequential:0x476fac0[24](1)
        _1000_Sequential:0x476fec0[28](1)
        _2052_Fix:0x47706e0[4](1)
        _1020_Sequential:0x4770ae0[8](1)
        _1016_Sequential:0x4770ee0[12](1)
        _1012_Sequential:0x47712e0[16](1)
        _1008_Sequential:0x47716e0[20](1)
        _1004_Sequential:0x4771ae0[24](1)
        _1000_Sequential:0x4771ee0[28](1)
        ...
```

図 16 特徴的なアクセスモデルの一例

データのグループが以降 14 回連続的に出現することが確認できた。この他にも、特徴的アクセスが確認できた。

5. まとめと今後の課題

メモリトレースから特定のルールに従って発見されるメモリアクセスパターンの連続区間やリピート区間をオンライン処理によって圧縮し、モデル化によって抽象度を向上させる手法の提案を行った。また、実駆動型アプリケーション解析ツールから得られた姫野ベンチマークの特定のループ処理のメモリアクセストレースを対象にモデル化を行い、約 95%のデータ数の圧縮率を達成した。今後の課題としては、姫野ベンチマークのメモリトレースのモデル化データを解析した際に、より複雑で特徴的なアクセス傾向が発見されたため、パターンモデルの抽象化を更に進め、より柔軟なモデル化を可能とする手法の開発が考えられる。

参考文献

[1] Sally A. McKee, ”Reflections on the memory wall”, Proceedings of the 1st conference on Computing frontiers (CF ’04), pp.162, 2004.

[2] Yukinori Sato, Hiroko Midorikawa and Toshio Endo, ”Identifying Working Data Set of Particular Loop Iterations for Dynamic Performance Tuning”, 6th Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT2013). Held in conjunction with the 40th Int’l Symposium on Computer Architecture (ISCA-40).