ハードウェア・カウンターを使用したプロファイルに基づく最適化 (HWPGO) によりパフォーマンスを大幅に向上

この記事は、インテルのウェブサイトで公開されている「Boost Performance with Hardware Counter Assisted Profile Guided Optimization (HWPGO)」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

HWPGO とは?

- 従来のインストルメンテーション・ベースの PGO の代替となるインテルのサンプルベースのプロファイルに基づく最適化 (SPGO) は、使用するメカニズムからハードウェア PGO (HWPGO) と呼ばれています。HWPGO は、最新のインテルの CPU で利用できるハードウェア・パフォーマンス・モニタリング・カウンターを活用します。
- パフォーマンス・モニタリング・カウンター (PMC) は、ハードウェア・イベントに関する情報を提供します。Linux の perf やインテル® VTune™ プロファイラーの SEP などのプロファイル・ツールは、コンパイラーで使用可能なこれらのイベントのサンプルベースのプロファイルを提供します。
- これにより、低オーバーヘッドで、迅速かつ効率良い方法で、ワークロード実行パスのパフォーマンス最適化プロファイルを取得できます。

パフォーマンス・モニタリング・カウンター (PMC)、またはハードウェア・パフォーマンス・カウンターは、CPU に組込まれた特殊なハードウェア・レジスターで、プログラムの実行中に発生するさまざまなパフォーマンス関連イベントを監視するように設計されています。これらのカウンターは、プロセッサーのパフォーマンス・モニタリング・ユニット (PMU) の一部です。

PMC は、次のようなさまざまなイベントを追跡できます。

リタイアした命令: プロセッサーによって正常に実行された命令の数。

キャッシュミス: データまたは命令がプロセッサーのキャッシュに見つからず、上位レベルのメモリーからフェッチする必要がある回数。

分岐予測ミス: プロセッサーが分岐命令の誤った結果を予測した回数。

浮動小数点演算: プロセッサーによって実行された浮動小数点演算の数。

メモリーアクセス: 読み取りおよび書き込み操作の数など、メモリーアクセスに関する情報。

ハードウェア割り込み: 受信した割り込みの数など、ハードウェア割り込みに関連するイベント。

収集オーバーヘッドの軽減

サンプルベースの PGO (SPGO) の一種である HWPGO は、従来のインストルメンテーション・ベースの PGO に比べて、プロファイルのオーバーヘッドを大幅に軽減します。通常、実稼働バイナリーを使用して実稼働ワークロードで収集を実行できるほどオーバーヘッドが小さくなるため、特別な「トレーニング」ワークロードを作成する必要性がなくなり、収集を簡素化します。

インストルメンテーションがユーザー空間で継続的に集計を実行するのに対して、HWPGO はイベントが N 回発生するごとにサンプリングするように PMU カウンターをプログラムします。そのため、オンライン集計の大部分を専用のハードウェアで行います。オーバーヘッドは通常 1% 程度ですが、サンプリング頻度を下げることでさらに軽減できます。各サンプルは最終分岐レコード (LBR) をキャプチャーして、効率良く命令トレースを形成します。

新しいフィードバック機能

オーバーヘッドの軽減だけでなく、HWPGO はインストルメンテーションでは不可能な新しいタイプのフィードバックを提供にします。例えば、PMU ハードウェアは、ハードウェアが分岐予測ミスしたときにサンプリングして、分岐予測ミス・プロファイルを作成できます。バージョン 2024.0 以降のインテル® コンパイラーは、このようなプロファイルを使用してコード生成を改善し、条件移動命令 (`cmov`) をより積極的に使用できます。

将来的には、PMU フィードバックの種類を増やすとともに、開発者が複数の PMU プロファイルを管理できるようにツールを改善することが検討されています。

分岐予測ミス・フィードバック

分岐予測ミスは、条件分岐命令 (if-then-else 文、ループ、プログラム中の意思決定を伴うコードなど) の結果をプロセッサーが誤って予測した場合に発生します。プロセッサーは分岐を予測すると、その予測に基づいて次の命令を推測実行しようとします。予測が間違っていた場合、プロセッサーは投機的な結果を破棄し、正しい命令から実行しなければなりません。この場合、命令のフェッチと実行にかかった時間とリソースが無駄になり、パフォーマンスの低下とパイプラインの非効率を招きます。予測ミス比率を理解するには、予測ミスと実行頻度のプロファイルを組み合わせる必要があります。コンパイラーやランタイムシステムは、実行頻度フィードバックと呼ばれるパフォーマンス最適化手法を使用して、異なるコードパスやプログラムセグメントの実行頻度に基づいて最適化を動的に調整します。実行頻度フィードバックの例については、こちらのドキュメント (英語) を参照してください。

このセクションでは、次のサンプル・ソースコードを使用します。
https://github.com/tcreech-intel/hwpgo-mispredict-example (英語)

分岐予測ミス・フィードバックの例

次のループについて考えてみましょう。

```
for (int i = 0; i < N; i++) {
  int *p;
  if(s1[i] > 8000) {
    p = &s2[i];
    int z = i * i * i * i * i * i * i;
    nop(p, z);
  } else {
    p =  &s3[i];
    nop(p, 3);
  }
  dst[i] = *p;
}
```

`if` 文の条件は `s1` 配列の内容に依存するため、コンパイラーの静的解析では、ハードウェアがどの程度正確に予測するかを推測するのは困難です。

予測が正しい場合は、条件ジャンプが適切ですが、予測が正しくない場合は、コンパイラーが制御フローを排除したほうがよいでしょう。次のループは同等ですが、条件式を使用して制御フローを排除しています。

```
for (int i = 0; i < N; i++) {
  int *p;

  int c = (s1[i] > 8000);
  p = c ? (&s2[i]) : (&s3[i]);
  int z = i * i * i * i * i * i * i;
  int arg2 = c ? z : 3;
  nop(p, arg2);

  dst[i] = *p;
}
```

この変換のデメリットは、条件 `c` が偽の場合でも `z` が計算されてしまうことです。言い換えれば、制御フローを排除するコストとして、`z` を投機的に計算しなければなりません。この変換の有効性は、`c` が正しく予測されない場合に高まります。

幸い、HWPGO を使用すると、ハードウェアの実際の分岐予測結果に基づいて、コンパイラーがこの判断を下すことができます。

このセクションの残りの部分では、この例に HWPGO を適用する手順を説明します。

分岐予測ミス・フィードバックのコンパイルフェーズ 1

最初に、PMU ベースのプロファイルに適したバイナリーを作成します。バイナリー内の命令とソース位置を関連付けるため、デバッグ情報が含まれていなければなりません。バイナリーは完全に最適化されていてもかまいません。

注: 汎用の `-gsplit-debug` 機能と DWARF パッケージファイルを使用して、バイナリーからデバッグ情報を分割できます。

必須ではありませんが、`-fprofile-sample-generate` オプションを使用すると、有用なデバッグ情報を確実に生成できます。次のように、サンプルをコンパイルします。

```
$ icx -O3 -fprofile-sample-generate unpredictable.c nop.c -o unpredictable
```

生成されたバイナリーの実行結果をメモしておきます。

```
$ time ./unpredictable
./unpredictable  1.04s user 0.00s system 99% cpu 1.050 total
```

バイナリーからデバッグ情報を削除したり、`-fprofile-sample-generate` オプションなしで再コンパイルした場合、パフォーマンスとコードの逆アセンブリー結果は同じになります。

分岐予測ミス・フィードバックのプロファイル収集

PMU プロファイルを生成することで、実行可能ファイルの動作についてハードウェアがどのような情報を提供できるかを詳しく見てみましょう。

- **Linux の場合:**

  ```
  $ perf record -b -e
  BR_INST_RETIRED.NEAR_TAKEN:uppp,BR_MISP_RETIRED.ALL_BRANCHES:upp -c 1000003
  -- ./unpredictable
  ```

  perf ツールは、Linux システムで利用可能なパフォーマンス解析およびプロファイル・ツールです。

  perf record: このコマンドは、perf を使用してパフォーマンス・イベントの記録を開始します。

  -b: このオプションは、各サンプルで最終分岐レコード (LBR) を収集するように perf に指示します。

  -e: このオプションは、プロファイルするイベント (BR_INST_RETIRED.NEAR_TAKEN と BR_MISP_RETIRED.ALL_BRANCHES) を指定します。

  BR_INST_RETIRED.NEAR_TAKEN:uppp: このイベントは、実行された分岐をカウントします。uppp 修飾子は、イベントがユーザーモードでのみ、最高精度と正確な分布でカウントされるべきであることを示します。

  BR_MISP_RETIRED.ALL_BRANCHES:upp: このイベントは、分岐予測ミスをカウントします。uppp 修飾子は、このイベントもユーザーモードでのみ、最高精度と正確な分布でカウントされるべきであることを示します。

  -c 1000003: このオプションは、perf がパフォーマンス・イベントをサンプリングする間隔を指定します。1000003 に設定すると、perf は監視イベントが 1,000,003 回発生するごとにサンプルを収集します。

-- ./unpredictable: コマンドのこの部分では、プロファイル対象のプログラムを指定します。./unpredictable は、実行可能なアプリケーションのパスです。

指定されたプログラム (./unpredictable) が実行されている間、perf ツールを使用してパフォーマンス・データを記録します。特に、near-taken 分岐命令のリタイアを監視し、1000003 のサンプリング間隔でイベントデータを記録します。記録されたデータは perf.data ファイルに保存され、後で解析することができます。

- **Windows でインテル® VTune™ プロファイラーの SEP を使用する場合:**

```
$ sep -start -out unpredictable.tb7 -ec
BR_INST_RETIRED.NEAR_TAKEN:PRECISE=YES:SA=1000003:pdir:lbr:USR=YES,BR_MISP_
RETIRED.ALL_BRANCHES:PRECISE=YES:SA=1000003:lbr:USR=YES -lbr no_filter:usr
-perf-script event,ip,brstack -app .\unpredictable
```

注: `-perf-script` オプションを指定すると、SEP は tb7 データに加えて単純なテキスト形式の perf.data.script ファイルを出力します。プロファイル生成には perf.data.script 出力のみが使用されます。

SEP は、インテルのパフォーマンス・ライブラリーとツールを使用します。

sep: インテル® VTune™ プロファイラーのコマンドライン・インターフェイス用のコマンドです。

-start: このオプションは、プロファイルを開始します。

-out unpredictable.tb7: このオプションは、プロファイル・データを保存する出力ファイルを指定します。ここでは、出力ファイルの名前は unpredictable.tb7 です。

-ec: このオプションは、プロファイルするイベント (BR_INST_RETIRED.NEAR_TAKEN と BR_MISP_RETIRED.ALL_BRANCHES) を指定します。

BR_INST_RETIRED.NEAR_TAKEN:PRECISE=YES:SA=1000003:pdir:lbr:USR=YES:
このイベントは、実行された分岐をカウントします。オプション **PRECISE=YES、SA=1000003、pdir、lbr、および USR=YES** は、それぞれ正確なイベントカウント、サンプリング間隔、正確な分布、最終分岐レコード、およびユーザーモード・イベントに関連する追加の設定パラメーターを提供します。

BR_MISP_RETIRED.ALL_BRANCHES:PRECISE=YES:SA=1000003:lbr:USR=YES:
このイベントは、分岐予測ミスをカウントします。オプション **PRECISE=YES、SA=1000003、lbr、および USR=YES** は、それぞれ正確なイベントカウント、サンプリング間隔、最終分岐レコード、およびユーザーモード・イベントに関連する追加の設定パラメーターを提供します。

-lbr no_filter:usr: このオプションは、ユーザーモード・イベントに対して、フィルターを適用せずに最終分岐レコード (LBR) スタック情報を収集するようにプロファイラーを設定します。

-perf-script event,ip,brstack: このオプションは、「perf.script」出力ファイルに含めるフィールドを指定します。ここでは、イベント名、命令ポインター (IP)、および分岐スタック情報が含まれています。

-app .\unpredictable: プロファイルするアプリケーションを指定します。ここでは、実行可能ファイル *unpredictable* への相対パスを指定しています。

要約すると、sep コマンドはインテル® VTune™ プロファイラーを起動します。そして、リタイアした near-taken 分岐命令、最終分岐レコード (LBR) スタック情報、および命令ポインター (IP) に関連するパフォーマンス・データを収集するように設定します。収集されたデータは、詳しく解析するためファイル unpredictable.tb7 に保存されます。

`-c 1000003` または `SA=1000003` を使用して、各イベントが 1,000,003 回発生するごとにサンプリングしました。サンプリング間隔が短いほど、サンプリング頻度が高くなります。頻度が高いほど、プロファイルの忠実度は高くなりますが、オーバーヘッドも高くなり、ストレージ要件も大きくなります。プログラムの特性に応じて、オーバーヘッドと忠実度のバランスを取るためサンプリング間隔を調整するとよいでしょう。プログラムの実行パターンとのエイリアシングを回避するため、素数のサンプリング間隔が推奨されます。

`llvm-profgen` を 2 回呼び出して (PMU プロファイル・タイプごとに 1 回ずつ)、ソースレベルのプロファイルを生成できます。この例では、人間が可読可能なプロファイルを生成するため `--format text` を使用していますが、そうでない場合はデフォルトのバイナリー形式を使用すべきです。

最初の呼び出しでは、LBR を使用してコード実行頻度のプロファイルを生成します。2 回目の呼び出しでは、サンプリングされた命令ポインターに基づいて、分岐予測ミスのプロファイルを生成します。

- **Linux:**

```
llvm-profgen --format text --output=unpredictable.freq.prof --
binary=unpredictable --sample-period=1000003 --perf-
event=BR_INST_RETIRED.NEAR_TAKEN:uppp --perfdata=unpredictable.perf.data
llvm-profgen --format text --output=unpredictable.misp.prof --
binary=unpredictable --sample-period=1000003 --perf-
event=BR_MISP_RETIRED.ALL_BRANCHES:upp --leading-ip-only --
perfdata=unpredictable.perf.data
```

- **Windows:**

```
llvm-profgen --format text --output=unpredictable.freq.prof --
binary=unpredictable --sample-period=1000003 --perf-
event=BR_INST_RETIRED.NEAR_TAKEN:pdir --
perfscript=unpredictable.perf.data.script
llvm-profgen --format text --output=unpredictable.misp.prof --
binary=unpredictable --sample-period=1000003 --perf-
event=BR_MISP_RETIRED.ALL_BRANCHES --leading-ip-only --
perfscript=unpredictable.perf.data.script
```

Windows の `llvm-profgen` コマンドは、SEP のテキスト形式の `-perf-script` 出力形式とさまざまなイベント名を使用するように調整されています。

最終分岐レコード (LBR) は実行頻度プロファイルにのみ使用されるため、予測ミス・プロファイルは `--leading-ip-only` を使用して生成する必要があります。

結果として得られる実行頻度プロファイル:

```
unpredictable:12905587092:0
 3.1: 202483466
 3.2: 202483466
 5: 202483466
 6: 119064278
 7: 119064278
 8: 124612654 nop:124612654
 11: 87128858 nop:87128858
 13: 202483466
nop:194225418:211741512
 1: 194225418
```

プロファイルの最初の列は関数の先頭からのソース行オフセットを示します。上記の C コードリストは行オフセット 3 から始まることに注意してください。2 番目の列は実行回数の推定値です。追加の列がある場合は、関数呼び出し回数を示します。

例えば、ソースコードの 5 行目にある「unpredictable」関数への分岐は、約 200,000,000 回実行されました。ソースでこれを確認できます (ループ本体の実行回数 `N` = 200,000,000 回)。

また、分岐の if/else ケースに対応する 2 つの `nop` 呼び出しが実行された割合は、それぞれ約 60% と 40% であることも分かります。これは分岐予測の精度を示すものではありません。例えば、約 50% の割合で分岐しても、予測精度は高い場合があります (この例については、predictable.c を参照してください)。

最終的に分岐予測が正しかったかどうかは、分岐予測ミス・プロファイルで確認できます。

```
unpredictable:172000516:0
 3.1: 0
 3.2: 0
 5: 86000258
 6: 0
 7: 0
 8: 0
 11: 0
 13: 0
 15: 0
```

およそ 200,000,000 回の反復のうち、86,000,258 回は誤って予測されていることから、ハードウェアにとってこの分岐の予測は非常に難しいことを示しています。このプログラムの分岐予測ミスは、実質的にすべてこの 1 つの分岐で発生するため、プロファイルは行わずに単純にカウントする `perf stat` を使用してこの結果を再確認できます。

```
$ perf stat -e br_misp_retired.all_branches -- ./unpredictable
 Performance counter stats for &apos;./unpredictable&apos;:

         85,739,375      br_misp_retired.all_branches

        1.025948268 seconds time elapsed
```

実際には、85,739,375 回の分岐予測ミスが発生しています。

一般的な llvm-profgen の問題

oneAPI `llvm-profgen` を使用することが重要です。PATH に含める場合は、oneAPI 環境のセットアップ時に `--include-intel-llvm` オプションを使用します。そうでない場合は、`icx --print-prog-name=llvm-profgen` を使用して適切なパスを見つけることができます (Windows では `icx /nologo /clang:--print-prog-name=llvm-profgen` を使用します)。

いくつかの警告が表示されることはよくありますが、実行頻度プロファイルを生成する際は、「`No samples in perf script!` (perf スクリプトにサンプルがありません!)」に注意すべきです。これは、PMU プロファイルが空か、間違ったバイナリーがプロファイルされたか、`--perf-event` で間違った PMU イベントが指定されたことを意味します。

分岐予測ミス・プロファイルの生成時にプロファイルが空になることはあるため、その場合は「`No samples in perf script!` (perf スクリプトにサンプルがありません!)」を無視できます。

分岐予測ミス・フィードバックのコンパイルフェーズ 2

予測ミス・プロファイルをコンパイラーに提供して再コンパイルすると、コンパイラーはより積極的に推測し、分岐を完全に回避できます。

```
$ icx -O3 -fprofile-sample-use=unpredictable.freq.prof -mllvm -unpredictable-hints-file=unpredictable.misp.prof unpredictable.c nop.c -o unpredictable.hwpgo
```

実行頻度プロファイルと分岐予測ミス・プロファイルの両方がコンパイラーに提供されます。コンパイラーは、この 2 つのプロファイルを組み合わせて使用し、個々の分岐予測ミス率を推定します。

この例では分岐予測ミス・フィードバックに注目していますが、実行頻度プロファイルはそれ自体でも役立ち、インストルメンテーション・ベースの PGO によって提供される基本ブロック実行カウントに似た情報を提供します。

分岐予測ミス・フィードバックの評価

分岐予測ミス・フィードバックを提供することで、実行時間とハードウェア・メトリックの両方でパフォーマンスが向上していることが分かります。

HWPGO 適用前:

```
$ perf stat -e
cycles:u,instructions,br_inst_retired.all_branches:u,br_misp_retired.all_branches
-- ./unpredictable

  Performance counter stats for &apos;./unpredictable&apos;:

     3,243,043,047      cycles:u
     3,619,535,187      instructions:u       #   1.12  insn per cycle
       917,083,309      br_inst_retired.all_branches:u
        85,966,707      br_misp_retired.all_branches:u

       1.021617622 seconds time elapsed
```

HWPGO 適用後:

```
$ perf stat -e
cycles:u,instructions,br_inst_retired.all_branches:u,br_misp_retired.all_branches
-- ./unpredictable.hwpgo

  Performance counter stats for &apos;./unpredictable.hwpgo&apos;:

     1,715,030,113      cycles:u
     4,000,954,710      instructions:u       #   2.33  insn per cycle
       600,132,829      br_inst_retired.all_branches:u
            13,322      br_misp_retired.all_branches:u

       0.541091594 seconds time elapsed
```

ここでは、実行時間が 1.8 倍 (1.02 秒から 0.54 秒に) 高速化されていますが、これは一般的ではありません。

また、if/else が条件ジャンプではなく、条件移動として実装されているため、実行される分岐の合計数が減っていることが分かります。

その結果、分岐予測ミスの数が大幅に減ります。代わりに、実行される命令の総数は増えますが、この例では予測ミスをなくすことでサイクルあたりの命令数 (IPC) がほぼ 2 倍になるため、有益です。

サンプル・ソースコード

この記事で説明したループのソースコード (unpredictable.c) は、次のリポジトリーから入手できます。

https://github.com/tcreech-intel/hwpgo-mispredict-example (英語)

このリポジトリーには、`cmov` が有害となる同様の例もあります。

アプリケーションでのハードウェア・プロファイルに基づく最適化の使用

ハードウェア・パフォーマンス・モニタリング・カウンターを使用したプロファイルに基づく最適化 (HWPGO) は、ワークロードの主要な実行パスを優先してパフォーマンスを最適化するオーバーヘッドの少ないアプローチを開発者に提供します。

これまでにプロファイルに基づく最適化 (PGO) を使用しており、より効率的な方法を探している場合は、スタンドアロン版またはインテル® oneAPI ベース・ツールキットの一部として利用可能な最新のインテル® oneAPI DPC++/C++ コンパイラーを今すぐチェックしてください。

関連情報

- インテル® oneAPI DPC++/C++ コンパイラー・ドキュメント (英語)
- プロファイルに基づく最適化 (PGO) (英語)
- インテル® oneAPI DPC++/C++ コンパイラー
- インテル® oneAPI ベース・ツールキット
- インテル® HPC ツールキット

製品および性能に関する情報

[1] 性能は、使用状況、構成、その他の要因によって異なります。詳細については、http://www.intel.com/PerformanceIndex/ (英語) を参照してください。