インテル® ソフトウェア開発ツールで始める
インテル® Inspector を使用した最適化

応用編

2019 年 4 月
エクセルソフト株式会社
テクニカルサポート - 竹田 賢人

本日の予定

10:30 – 17:00

• コードの現代化と最適化	10:30 – 11:00
• 並列アプリケーションのスケーラビリティーの問題を特定	11:00 – 12:00
昼食	
• インテル® コンパイラーによる高度な最適化	13:00 – 14:00
• インテル® VTune™ Amplifier 応用編	14:00 – 15:00
休憩	
• インテル® Advisor 応用編	15:15 – 16:15
• インテル® Inspector 応用編	16:15 – 16:45
• まとめと Q/A	16:45 – 17:00

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

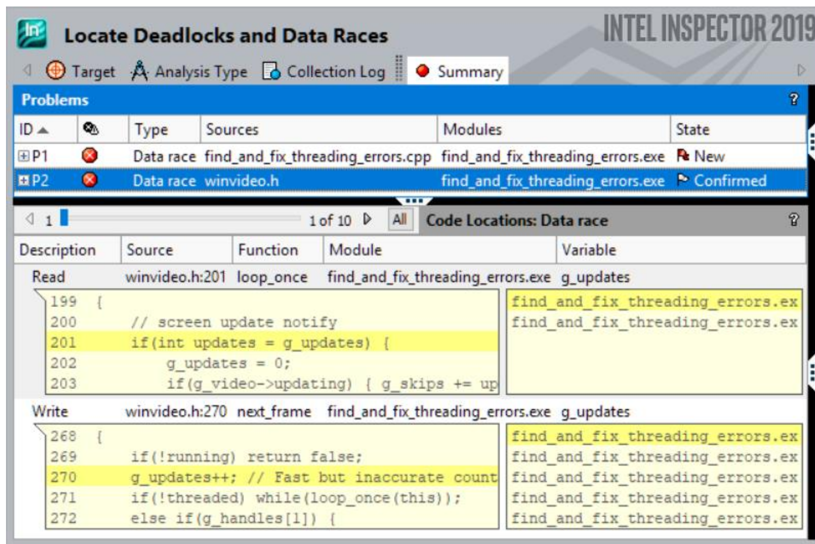
内容

- インテル® Inspector 2019 概要と新機能
- インテル® Inspector の不揮発性インスペクターの使用例

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® Inspector によるメモリーとスレッドのデバッグ

メモリーリーク、異常、データ競合、デッドロックの特定とデバッグ



正当性検証ツールにより ROI が 12%-21%[1] 向上

- 早期に問題を発見したほうが修正コストが少なくて済む
- 競合やデッドロックは簡単に再現できない
- メモリーエラーをツールなしで発見するのは困難

デバッガー統合により迅速な診断が可能

- 問題の直前にブレークポイントを設定
- デバッガーで変数とスレッドを確認

バージョン 2019 の新機能

不揮発性メモリーエラーを発見

- 不足している/冗長なキャッシュフラッシュ
- ストアフェンスの不足
- アウトオブオーダーの不揮発性メモリーストア
- PMDK トランザクションの Redo (やり直し) ログエラー

詳細: isus.jp/intel-inspector-xe/

[1] **コスト要因 - Square Project による分析**
CERT: U.S. Computer Emergency Readiness Team および Carnegie Mellon CyLab
NIST: National Institute of Standards & Technology : Square Project の結果

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

バージョン 2019 の新機能
インテル® Inspector

電源をオフにしてもデータを保持するには?

- キャッシュから不揮発性メモリーへのフラッシュが必要

不揮発性メモリーエラーを発見

- 不足している/冗長なキャッシュフラッシュ
- ストアフェンスの不足
- アウトオブオーダーの不揮発性メモリーストア
- PMDK トランザクションの Redo (やり直し) ログエラー

Problems

ID		Type	Sources	Modules	State
P1		Missing cache flush	MSVCR120D.dll:0x4B...	MSVCR120D.dll	New
P2		Missing cache flush	trace.pmem.cpp:201...	tachyon.exe	New
P3		Missing cache flush before unmap()	pmem_windows.cpp...	tachyon.exe	New

1 1 of 1 All Code Locations: Missing cache flush

Description	Source	Function	Module	Variable
Controlled variable	trace.pmem.cpp:243	do_render	tachyon.exe	
Unflushed memory store	trace.pmem.cpp:242	do_render	tachyon.exe	

```
241            color_t *pixel = &rb[width * y + x];
242            *pixel = render_one_pixel(x, y, local
243            rs->pixels_stored++;
244            drawing.put_pixel(*pixel);
245        }
```

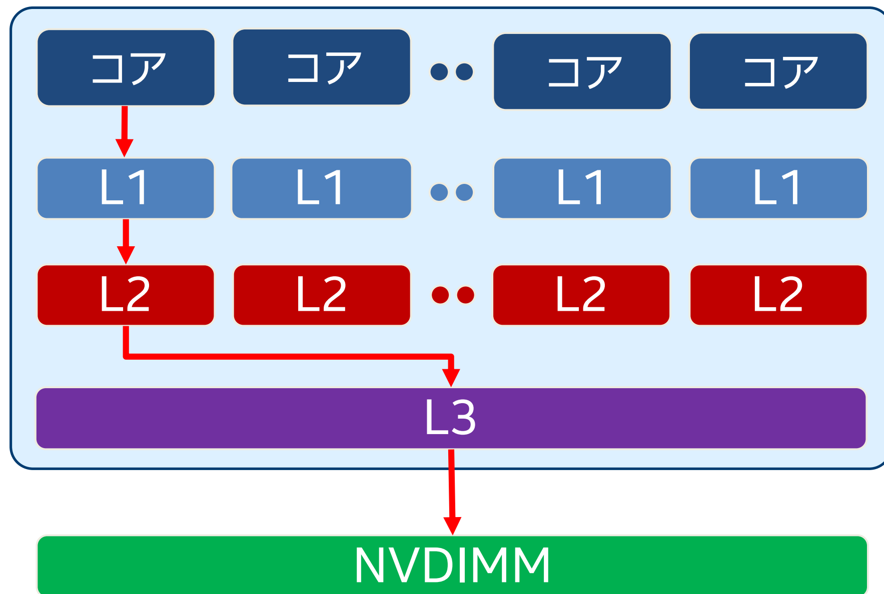
```
240        {
241            color_t *pixel = &rb[width * y + x];
242            *pixel = render_one_pixel(x, y, local
243            rs->pixels_stored++;
244            drawing.put_pixel(*pixel);
```

PMDK = Persistent Memory Developer Kit (不揮発性メモリー開発キット、旧 NVML)

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

不揮発性メモリー利用時のメモリー階層

- 不揮発性メモリーは、アプリケーションのパフォーマンスと信頼性を向上させる大きな可能性がある、新しいメモリー・ストレージ・テクノロジーです
- NVDIMM にあるデータは、バイトアドレス指定可能で、システムやプログラムがクラッシュしても保持されます。NVDIMM のアクセス・レイテンシーは、DRAM に匹敵します
- プログラムは、通常の CPU ロード/ストア命令を使用して NVDIMM を読み書きします



ただし、コードのパフォーマンスを最大限に引き出すため、開発者はいくつかのプログラミングの課題に対処する必要があります

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

不揮発性メモリー利用時の問題

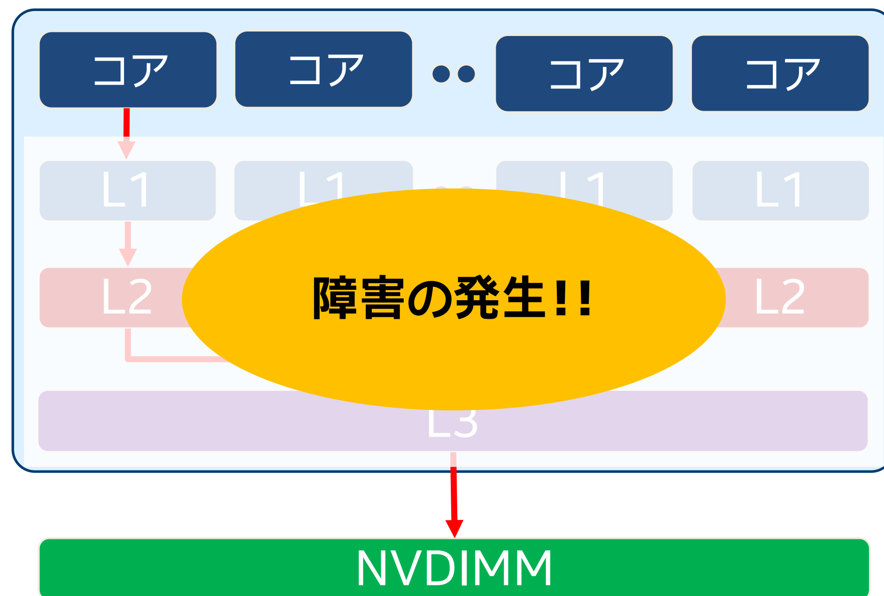
不揮発性メモリーへのストアはキャッシュの影響により直ちに永続化されません

データは、不揮発性メモリーに達するまで永続ではありません

不揮発メモリーへの格納順序とストア順序は異なる場合があります

揮発性キャッシュのデータは電源停止時に消失します

NVDIMM を使用するメモリー階層



https://www.isus.jp/products/inspector/detect-persistent-memory-errors-with-persistence-inspector/

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

不揮発性メモリーの利用例

次のコード例について考えてみます。
名前、住所、入力フラグを格納する address 構造体を扱います。

```
int main()
{
    struct address *head = NULL;
    int fd;
    fd = open("addressbook.pmem", O_CREAT|O_RDWR, 0666);
    posix_fallocate(fd, 0, sizeof(struct address));

    head = (struct address *)mmap(NULL, sizeof(struct address), PROT_READ|PROT_WRITE,
      MAP_SHARED, fd, 0);
    close(fd);

    strcpy(head->name, "Clark Kent");
    strcpy(head->address, "344 Clinton St, Metropolis, DC 95308");
    head->valid = 1;

    munmap(head, sizeof(struct address));

    return 0;
}
```

```
struct address {
    char name[64];
    char address[64];
    int valid;
}
```

munmap を呼び出す前に電力が失われた場合、

- head->name、head->address、および head->valid のいずれも、メモリーシステムに到達せず、永続化されない
- 3 つすべてが永続化される
- 1 つまたは 2 つのみが永続化される

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

NVDIMM を活用しつつ、データの一貫性と回復可能な状態を維持するために…

キャッシュをフラッシュするタイミングは?

- データはキャッシュから不揮発性メモリーへのフラッシュが必要
- しかし大量のフラッシュはパフォーマンスの低下を招く

キャッシュをフラッシュする順序は?

- 「head->valid」を設定した後に、「 head->name 」や「 head->address 」のフラッシュは正しいか

データはどの順序でストアするべきか

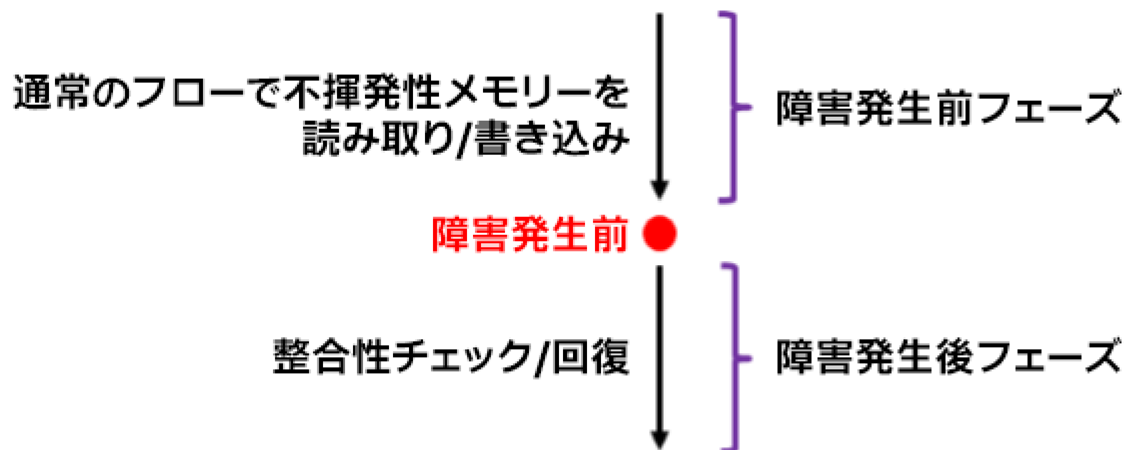
- 「 head->name 」や「 head->address 」をストアする前に「head->valid」の設定は正しいか

PMDK = Persistent Memory Developer Kit (不揮発性メモリー開発キット、旧 NVML)

不揮発性メモリー･アプリケーションの動作

通常、障害によって不揮発性メモリー･アプリケーションは、障害発生前の実行と、障害発生後の実行の 2 つのフェーズに分けることができます

■ 障害発生前フェーズ
■ 障害発生後フェーズ



© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® Inspector の不揮発性インスペクターの使用フロー

- 環境の設定
- アプリケーションの準備
- インテル® Inspector の不揮発性インスペクターに障害発生後フェーズを通知
- 障害発生前フェーズを解析する
- 障害発生後フェーズを解析する
- 検出された問題をレポートする
  - キャッシュフラッシュの不足
  - 冗長または不要なキャッシュフラッシュ
  - アウトオブオーダーの不揮発性メモリーストア
  - 取り消しログがない更新
  - 更新がない取り消しログ

環境の設定
アプリケーションの準備
障害発生前解析の実行
アプリケーションが PMDK トランザクション・アプリケーションであるか、または PMDK 取り消しログのチェックのみが重要?
いいえ
はい
障害発生後解析の実行
レポート

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

環境設定とアプリケーションの準備

指定したディレクトリー (例えば /opt/intel/) に Parallel Studio XE ツールのファイルがインストールされたら、インテル® Inspector の inspxe-vers.sh を実行してパスを追加します。

```
$ Source /opt/intel/inspector/inspxe-vars.sh
```

前述のとおり、不揮発性メモリー・アプリケーションは通常 2 つのフェーズで構成されます。インテル® Inspector の不揮発性インスペクターを使用するには、これらの 2 つのフェーズのコードを識別する必要があります

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

利用例を不揮発性インスペクターで解析する

```
struct address {
char name[64];
char address[64];
int valid;
};
```

障害発生前フェーズ

```
int main()
{
    struct address *head = NULL;
    int fd;

    fd = open("addressbook.pmem", O_CREAT | O_RDWR, 0666);
    posix_fallocate(fd, 0, sizeof(struct address));

    head = (struct address *)mmap(NULL, sizeof(struct address),
              PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    close(fd);

    strcpy(head->name, "Clark Kent");
    strcpy(head->address, "344 Clinton St, Metropolis, DC 95308");
    head->valid = 1;

    ...

}
```

障害発生後フェーズ

```
int main()
{
    struct address *head = NULL;
    int fd;

    fd = open("addressbook.pmem", O_CREAT | O_RDWR, 0666);
    posix_fallocate(fd, 0, sizeof(struct address));

    head = (struct address *)mmap(NULL, sizeof(struct address),
              PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    close(fd);

    if(head->valid==1)
        printf("%s%s¥n", head->name, head->address);

    ...

}
```


© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

障害発生前後フェーズを解析する

次のコマンドは、障害発生前フェーズ解析を実行します:

$ pmeminsp cb [options] -- <application> <application-arguments]

次のコマンドは、障害発生後フェーズ解析を実行します:

$ pmeminsp ca [options] -- <application> <application-arguments]

不揮発性メモリーファイルにより操作している場合は、
-pmem-file <pmem-file> オプションを pmeminsp コマンドに指定する

cb : check-before-unfortunate-event
ca : check-after-unfortunate-event

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

検出された問題をレポートする

検出された不揮発性メモリーの問題のレポートを生成するには、次のコマンドを実行します:

$ pmeminsp rp [options] -- <before-unfortunate-event-application> [after-unfortunate-event-application]

インテル® Inspector の不揮発性インスペクターによって生成される診断例:

- キャッシュフラッシュの不足
- 冗長または不要なキャッシュフラッシュ
- アウトオブオーダーの不揮発性メモリーストア
- 取り消しログがない更新
- 更新がない取り消しログ

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

レポート例:キャッシュフラッシュの不足

不揮発性メモリーストア (最初のストア) のキャッシュフラッシュの不足は、常に後続の不揮発性メモリーストア (2 つ目のストア) に関連しています。潜在的な悪影響は、2 つ目のストアの後に障害が発生した場合、2 つ目のストアは永続化されますが、最初のストアは永続化されないことです

最初のメモリーストア

```
1  in /home/joe/pmeminsp/addressbook/writeaddressbook!main at writeaddressbook.c:24 - 0x6ED,
2  in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main, at: - 0x21F43
3  in /home/joe/pmeminsp/addressbook/writeaddressbook!_start at: - 0x594
```

これは、次に示す 2 つ目のメモリーストアの前にフラッシュされません

```
1  in /home/joe/pmeminsp/addressbook/writeaddressbook!main at: writeaddressbook.c:26 - 0x73F,
2  in /lib/x86_64-linux-gnu/libc.so.6!__libc_start_main, at: - 0x21F43,
3  in /home/joe/pmeminsp/addressbook/writeaddressbook!_start at: - 0x594
```

最初のストアの場所からのメモリーロードを以下に示します:
in /lib/x86_64-linux-gnu/libc.so.6!strlen at: - 0x889DA
これは、次に示す 2 つ目のストアの場所からのメモリーロードに依存します:
in /home/joe/pmeminsp/addressbook/readaddressbook!main at readaddressbook.c:22 - 0x6B0

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

コードの変更例

```
struct address {
char name[64];
char address[64];
int valid;
};
```

- CLFLUSHOPT 命令を使用してキャッシュフラッシュのタイミングを制御します
- CLFLUSHOPT 命令の後に正しい順序付けを強制するために、ストアフェンスを実行します
- C コンパイラは組み込み関数、_mm_clflushopt()、_mm_sfence()を提供します

```
int main()
{
    struct address *head = NULL;
    int fd;
    fd = open("addressbook.pmem", O_CREAT | O_RDWR, 0666);
    posix_fallocate(fd, 0, sizeof(struct address));

    head = (struct address *)mmap(NULL, sizeof(struct address),
PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    close(fd);

    strcpy(head->name, "Clark Kent");
    strcpy(head->address, "344 Clinton St, Metropolis, DC 95308");
    _mm_clflushopt(&(head->name));
    _mm_clflushopt(&(head->address));
    _mm_sfence();

    head->valid = 1;
    _mm_clflushopt(&(head->valid));
    _mm_sfence();

    ...

}
```

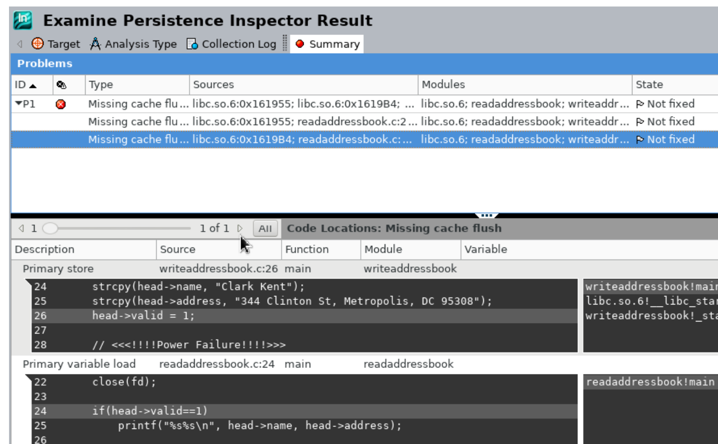

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® Inspector プロジェクトの作成

不揮発性インスペクターにより検出された不揮発メモリーに関するエラーは、インテル® Inspector の GUI から確認できます。

$ pmem report コマンドに
-insp オプションを指定すると、
インテル® Inspector プロジェクトを
作成します。

実行後、.pmeminspdata ディレクトリが作成され、以降の解析結果が連番で生成されます。



インテル® Inspector で不揮発性インスペクターの結果を開く
$ inspxe-gui .pmeminspdata/r000pmem/r000pmem.inspxe

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® Inspector の不揮発性インスペクターに障害発生後フェーズを通知

インテル® Inspector の不揮発性インスペクターには、アプリケーションが障害発生後フェーズ解析の開始と停止をランタイムにツールに通知する API セットがあります

```
1  #define PMEMINSP_PHASE_AFTER_UNFORTUNATE_EVENT 0x2
2  void __pmeminsp_start(unsigned int phase);
3  void __pmeminsp_pause(unsigned int phase);
4  void __pmeminsp_resume(unsigned int phase);
5  void __pmeminsp_stop(unsigned int phase);
```

例えば、障害発生後フェーズが関数 recover() 呼び出しの間である場合、単純に関数の入口に `__pmeminsp_start(PMEMINSP_PHASE_AFTER_UNFORTUNATE_EVENT)` を配置し、関数の出口に `__pmeminsp_stop(PMEMINSP_PHASE_AFTER_UNFORTUNATE_EVENT)` を配置します

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

インテル® Inspector の不揮発性インスペクターに障害発生後フェーズの開始と停止を通知する例

```
1   #include "pmeminsp.h"
2
3    …
4    … …
5
6   void recover(void)
7   {
8       __pmeminsp_start(PMEMINSP_PHASE_AFTER_UNFORTUNATE_EVENT);
9       …
10      __pmeminsp_stop((PMEMINSP_PHASE_AFTER_UNFORTUNATE_EVENT);
11  }
12
13  void main()
14  {
15      …
16      … = mmap(…..);
17      __pmeminsp_stop((PMEMINSP_PHASE_AFTER_UNFORTUNATE_EVENT);
18      …
19      recover();
20  }
```

インテル® Inspector の不揮発性インスペクターの API は、libpmeminsp.so ライブラリーで定義されています

アプリケーションをビルドする際に、正しいオプションを指定します。以下に例を示します:
-I /home/joe/pmeminsp/include -L /home/joe/pmeminsp/lib64 -lpmeminsp

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

まとめ

- 不揮発性メモリーは、魅力的な新しいテクノロジーです
- ここで説明したように、このテクノロジーにはいくつかのプログラミングの課題があります
- これらの課題は、プログラム・ライフサイクルの早期に問題を発見し、非常に大きな投資対効果が得られる、インテル® Inspector の不揮発性インスペクターなどのツールを使用することで克服できます

© 2019 iSUS *その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

iSUS