ベータ版インテル® C++ コンパイラーおよびベータ版インテル® Fortran コンパイラー向けの GPU への OpenMP* オフロード導入

この記事は、インテル® デベロッパー・ゾーンに公開されている「Get Started with OpenMP* Offload to GPU for the Intel® C++ Compiler and Intel® Fortran Compiler (Beta)」の日本語参考訳です。

インテル® C++ コンパイラーおよびインテル® Fortran コンパイラーの GPU への OpenMP* オフロード機能は、広範囲のアクセラレーター向けに OpenMP* ソースファイルをコンパイルできます。

はじめに

既知の問題と最新情報については、リリースノートをご覧ください。

- インテル® oneAPI C++ コンパイラー・リリースノート (英語)
- インテル® oneAPI HPC ツールキット向けインテル® Visual Fortran コンパイラー 19.1 (Windows* 版) (英語)

GPU 向けの OpenMP* 4.5 サブセット

oneAPI 向けインテル® C++ コンパイラー (英語) は、ユーザー定義のリダクション (UDR) を除く、CPU 向けのすべての OpenMP* 4.5 機能をサポートします。GPU とオフロードは、OpenMP* 4.5 および OpenMP* 5.0 の declare variant のサブセットをサポートします。この機能は、ハイパフォーマンス・コンピューティング (HPC) アプリケーションで使用される OpenMP* 構造の解析とコレクションに基づいています。サポートの詳細については、『oneAPI 向けインテル® C++ コンパイラー・デベロッパー・ガイドおよびリファレンス』の「OpenMP* のサポート」(英語) を参照してください。

OpenMP* 4.5 のサブセットに含まれる GPU ターゲットのデバイスサポート

以下は、GPU および CPU 向けの次世代コンパイラーでサポートされる OpenMP* プラグマです。

- `declare_target`
- `declare_simd`
- `target`
- `target_teams`
- `target_teams_distribute`
- `target_teams_distribute_simd`
- `target_parallel`
- `target_parallel_for`
- `target_parallel_for_simd`
- `target_teams_distribute_parallel_for`
- `target_teams_distribute_parallel_for_simd`
- `target_variant_dispatch`
- `simd`
- `master`
- `atomic`
- `barrier`
- `parallel_for`
- `teams`
- `teams_distribute`

- teams_distribute_parallel_for
- teams_distribute_parallel_for_simd
- target_data)
- target_update
- target_enter_data
- target_exit_data
- parallel
- single
- for
- for_simd
- parallel_for_simd
- distribute_parallel_for
- parallel_sections
- sections

以下は、CPU および GPU 向けの次世代コンパイラーでサポートされる OpenMP* 指示句です。

- if
- final
- num_threads
- safelen
- simdlen
- collapse
- default
- private
- firstprivate
- lastprivate
- shared
- reduction
- linear
- aligned
- proc_bind
- schedule
- ordered
- nowait
- untied
- mergeable
- flush
- read
- write
- update
- capture
- seq_cst
- depend
- device
- simd
- map
- num_teams
- thread_limit
- num_tasks
- ist_schedule
- defaultmap
- to
- from
- use_device_ptr
- is_device_ptr

以下は、CPU ホストで利用可能なランタイム・サポート・ルーチンです。

- `EXTERN int omp_get_num_devices(void);`
- `EXTERN int omp_get_initial_device(void);`
- `EXTERN void *omp_target_alloc(size_t size, int device_num);`
- `EXTERN void omp_target_free(void *device_ptr, int device_num);`
- `EXTERN int omp_target_is_present(void *ptr, int device_num);`
- `EXTERN int omp_target_memcpy(void *dst, void *src, size_t length, size_t dst_offset, size_t src_offset, int dst_device, int src_device);`
- `EXTERN int omp_target_memcpy_rect(void *dst, void *src, size_t element_size, int num_dims, const size_t *volume, const size_t *dst_offsets, const size_t *src_offsets, const size_t *dst_dimensions, const size_t *src_dimensions, int dst_device, int src_device);`
- `EXTERN int omp_target_associate_ptr(void *host_ptr, void *device_ptr, size_t size, size_t device_offset, int device_num);`
- `EXTERN int omp_target_disassociate_ptr(void *host_ptr, int device_num);`
- `EXTERN int omp_is_initial_device(void);`
- `EXTERN int omp_get_initial_device(void);`
- `EXTERN void kmp_global_barrier_init(void); // Intel externsion`
- `EXTERN void kmp_global_barrier(void); // Intel externsion`
- `EXTERN void omp_set_default_device(int dev_num )`
- `EXTERN int omp_get_default_device(void)`

以下は GPU 向けのデバイス・ランタイム・ルーチンです。

- `EXTERN int omp_get_team_num(void);`
- `EXTERN int omp_get_num_teams(void);`
- `EXTERN int omp_get_team_size(int);`
- `EXTERN int omp_get_thread_num(void);`
- `EXTERN int omp_get_num_threads(void);`
- `EXTERN int omp_in_parallel(void);`
- `EXTERN int omp_get_max_threads(void);`
- `EXTERN int omp_get_device_num(void);`
- `EXTERN int omp_get_num_devices(void);`

以下は環境変数です。

- `OMP_DEFAULT_DEVICE` で検出された制御デフォルトデバイス:
  - o 負ではない整数値を指定。
- `export OMP_TARGET_OFFLOAD={"MANDATORY" | "DISABLED" | "DEFAULT" }`:
  - **MANDATORY**: GPU またはアクセラレーターで実行されるターゲット領域のコード。
  - **DISABLED**: CPU で実行されるターゲット領域のコード。
  - **DEFAULT**: デバイスが利用可能な場合に GPU で実行され、利用できない場合は CPU にフォールバックされるターゲット領域のコード。

OpenMP* 4.5 (Gen9 以降のターゲットを含む) サポート

OpenMP* オフロードランタイムは、GPU ターゲット固有のマッピングを行います。OpenMP* 実行モデルから GPU ハードウェアへのマッピングを以下に示します。

OpenMP*	GPU ハードウェア	チーム数/スレッド数/SIMD レーン数
チーム	サブスライス (SS)	DSS ごとに 2+ チーム
スレッド	EU スレッド	DSS ごとに 64 EU スレッドを利用可能
SIMD	レーン (または "チャネル")	SIMD1、SIMD4、SIMD8、SIMD16、SIMD32

GPU サポートでは、複数レベルの並列処理が複数の OpenMP* チーム、スレッド、SIMD レーンを介して有効になり、すべてのレベルでハードウェア・リソースを完全に活用します。インテル® GPU では複数のチームをサブスライスにマッピングできるため、アプリケーションは DSS ごとに 64 以上のスレッドを実行できます。

- チーム内のスレッド間でハードウェア・バリアの使用が許可されます。
- OpenMP* スレッドのセマンティクスを許可します (独立した分岐など)。
- チーム間の同期が可能です。
- チームを使用してマシン全体を利用できます (より多くのマイグレーションが必要)。
- OpenMP* simd またはコンパイラーのベクトル化により SIMD を活用します。インテル® oneAPI HPC ツールキットでは、OpenMP* simd が CPU でもサポートされます。GPC では、カーネルレベルでコンパイラーのベクトル化を利用し、SIMT のような SIMD 実行モデルを活用します。MS66 では、インテル® oneAPI HPC ツールキットは GPU 上の OpenMP* simd もサポートします。明示的な SIMD ベクトル化モードでは、`afelen(n)` 句は安全性をチェックするためにのみ使用され、コンパイラーは `simdlen < n` を選択できます。`simdlen(n)` 句は SIMD 命令レーン/チャンネルにマッピングされます。

OpenMP* 3.0 レガシー・アプリケーションで Gen9 以降をターゲットにするオプションのサポート (以前の調査に基づきます)

2 つの新しいオプションがあります。

- `-fiopenmp`
- `-fopenmp-targets=spir64`

CPU および GPU での OpenMP* とオフロード実行をサポートします。`-fiopenmp` オプションは、LLVM* での OpenMP* 変換をサポートするミドルエンドを有効にします (clang フロントエンドではサポートされません)。`-fopenmp-targets=spir64` オプションを使用すると、コンパイラーは GPU デバイス向けのバイナリーとして `x86 + SPIR64` ファットバイナリーを生成できます。

C++ および Fortran の Gen9 以降のディレクティブのサポート強化

OpenMP* オフロードモデルを使用してライブラリー関数を記述し、オフロード領域をさらに高速なバージョンの関数にすることもできます。この利用モデルをサポートするため、コンパイラーは target variant dispatch 構造と拡張機能をサポートし、関数呼び出しの周辺でディスパッチ・コードを発行するようコンパイラーに指示します。構造はオプションとして device 句を受け入れます。構文は次のとおりです。

```
#pragma omp target variant dispatch [device( n )]
関数呼び出し
```

ディスパッチ・コードは、関数の基本バージョンと異形バージョンのどちらを呼び出すのかを決定するランタイム・チェック・ルーチンです。GPU が利用可能であれば異形バージョンが呼び出され、それ以外は基本バージョンが呼び出されます。device (*n*) 句が指定されると、デバイス *n* が利用可能な場合にのみ異形バージョンが呼び出されます。

関数の基本バージョンと異形バージョンの名前を指定するため、このコンパイラーのリリースでは次のように OpenMP* 5.0 declare variant 構造のサブセットをサポートしています。

```
#pragma omp declare variant ( variant_func )match(construct={target variant
dispatch}, device={arch(gen)})
```

ベース関数

以下はこの機能を使用した例です。

```
#include <stdio.h>

#define N 1024

float __attribute__((nothrow, noinline)) vecadd_gpu_offload() {
  float result = 0.0;
  float a[N], b[N];

  #pragma omp target parallel for reduction(+: result) map(to: a, b)
  for (int k=0; k<N; k++) {
    a[k] = k;
    b[k] = k + 1;
    result = result + a[k] + b[k];
  }
  printf("GPU version was called. ");
  return result;
}

#pragma omp declare variant(vecadd_gpu_offload) \
            match(construct={target variant dispatch}, device={arch(gen)})
float __attribute__((nothrow, noinline)) vecadd_base() {
  float result = 0.0;
  float a[N], b[N];

  #pragma omp parallel for reduction(+: result)
  for (int k=0; k<N; k++) {
    a[k] = k;
    b[k] = k + 1;
    result = result + a[k] + b[k];
  }
  printf("CPU version was called. ");
  return result;
}

int main() {
  float result=0.0;

  #pragma omp target variant dispatch
  {
     result = vecadd_base();
  }

  if (result == 1048576.0) {
    printf("PASSED: correct results\n");
    return 0;
  }

  printf("FAILED: incorrect results\n");
  return -1;
}
```

Gen9 以降のターゲット領域の制限に関するドキュメント

OpenMP* オフロードは GPU 向けの OpenCL* ランタイムスタック上にあるため、OpenCL* カーネル関数に適用される次の制限が OpenMP* オフロード領域のコードにもあてはまります。

- 再帰関数呼び出し (コンパイル時の定数式を除く)
- プレースメント以外の new と delete
- goto 文の制限
- register と thread_local ストレージ修飾子
- 仮想関数修飾子
- 関数ポインター (コンパイル時の定数式を除く)
- 仮想関数
- 例外処理
- C++ 標準ライブラリー (GPU では printf のみサポート)
- ラムダ式から関数への暗黙的なポインター変換
- 可変関数
- 可変長配列 (VLA) (タスクモデルと非同期オフロードではサポートされない)

OpenMP* オフロードの利用例

次のコードは、OpenMP* target、teams、distribute および parallel for を組み合わせた簡単な行列乗算の例です。

```
// matmul.cpp: OpenMP* のオフロードを使用した行列乗算の例
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define MAX 128
int A[MAX][MAX], B[MAX][MAX], C[MAX][MAX], C_SERIAL[MAX][MAX];

typedef int BOOL;
typedef int TYPE;

BOOL check_result(TYPE *actual, TYPE *expected, unsigned n) {
    for (unsigned i = 0; i < n; i++) {
        if(actual[i] != expected[i]) {
            printf("Value mismatch at index = %d. Expected: %d"
                   ", Actual: %d.\n", i, expected[i], actual[i]);
            return 0;
        }
    }
    return 1;
}

void __attribute__ ((noinline)) Compute()
{
  #pragma omp target teams distribute parallel for map(to: A, B) map(tofrom: C) \
                                                   thread_limit(128)
  {
    for (int i = 0; i < MAX; i++)
    for (int j = 0; j < MAX; j++)
    for (int k = 0; k < MAX; k++)
         C[i][j] += A[i][k] * B[k][j];
  }
}

int main() {
  for (int i = 0; i < MAX; i++)
    for (int j = 0; j < MAX; j++) {
      A[i][j] = i + j - 1;
      B[i][j] = i - j + 1;
    }

  for (int i = 0; i < MAX; i++)
  for (int j = 0; j < MAX; j++)
```

```
  for (int k = 0; k < MAX; k++)
       C_SERIAL[i][j] += A[i][k] * B[k][j];

  Compute();

  if (!check_result((int*) &C[0][0], (int*) &C_SERIAL[0][0], MAX * MAX)) {
    printf("FAILED\n");
    return 1;
  }

  printf("PASSED\n");
  return 0;
}
```

コンパイルと実行コマンド

注: Linux* では、ホストコードをコンパイルするため GCC 4.8.5 以降をインストールする必要があります。これは、C++ アプリケーション・バイナリー・インターフェイス (ABI) の変更による非互換性を回避するためです。

1. **コンパイル**: GPU オフロードを起動するため、`icx`、`icpx`、または `ifx` コンパイル・オプションを使用してソースコードをコンパイルします。

   ```
   icx -fiopenmp –fopenmp-targets=spir64 file.cpp matmul_offload.cpp -o matmul
   ```

   または

   ```
   icpx -fiopenmp –fopenmp-targets=spir64 file.cpp matmul_offload.cpp –o
   matmul
   ```

   または

   ```
   ifx -fiopenmp –fopenmp-targets=spir64 file.f90 matmul_offload.f90 –o matmul
   ```

2. **実行**: 環境変数を設定します: `export OMP_TARGET_OFFLOAD="MANDATORY"`。デフォルト値は `DEFAULT` で、CPU と GPU で実行できることを意味します。以下に例を示します。

   ```
   sh-4.2$ ./matmul
   ** Program Scope patch lists **
   ** Kernel Patch Lists : Kernel Name =
   __omp_offloading_811_600f59f6__Z7Computev_l14 **
   BINDING_TABLE_STATE
           Entry[   0 ]        : 00000000 (0)
           Entry[   1 ]        : 00000040 (2)
           Entry[   2 ]        : 00000080 (4)
   INTERFACE_DESCRIPTOR_DATA = { 00000000, 00000000, 00000000, 00000000,
   000000c3,
   00000000, 00000000, 00000003 }
   KernelStartPointer =               : 0
   Kernel64bitStartPointer =          : 0
   SoftwareExceptionEnable =          : 0
   MaskStackExceptionEnable =         : 0
   IllegalOpcodeExceptionEnable =     : 0
   FloatingPointMode =                : 0
   ThreadPriority =                   : 0
   SingleProgramFlow =                : 0
   DenormMode =                       : 0
   SamplerCount =                     : 0
   SamplerStatePointer =              : 0
   BindingTableEntryCount =           : 3
   BindingTablePointer =              : 6
   ConstantURBEntryReadOffset =       : 0
   ConstantURBEntryReadLength =       : 0
   ```

```
      NumberOfThreadsInThreadGroup =     : 0
      GlobalBarrierEnable =              : 0
      SharedLocalMemorySize =            : 0
      BarrierEnable =                    : 0
      RoundingMode =                     : 0
      CrossThreadConstantDataReadLength  : 3
      Kernel Name: __omp_offloading_811_600f59f6__Z7Computev_l14 PASSED
```

GPU 向けに最適化された LIBM 関数のコンパイラー統合の強化

場合によっては、数学関数には精度とパフォーマンスのトレードオフが異なる複数の異形があります。コンパイラーは、オプションによって適切な異形を選択する手段を提供します。コンパイラーでサポートされるすべての fp-model は、インテル® C++ コンパイラーとインテル® Fortran コンパイラーの OpenMP* の GPU へのオフロード機能でサポートされます。インテル® C++ コンパイラーでサポートされる fp-model も移行されます。以下は、OpenCL* 組込み数学関数に基づく Gen9 以降でサポートされる数学関数のリストです。

```
std::unordered_map<std::string, std::string> llvm::vpo::OCLBuiltin = {
    // float:
    {"sinf",                  "_Z3sinf"},
    {"cosf",                  "_Z3cosf"},
    {"tanf",                  "_Z3tanf"},
    {"erff",                  "_Z3erff"},
    {"expf",                  "_Z3expf"},
    {"logf",                  "_Z3logf"},
    {"log2f",                 "_Z4log2f"},
    {"powf",                  "_Z3powff"},
    {"sqrtf",                 "_Z4sqrtf"},
    {"fmaxf",                 "_Z4fmaxff"},
    {"llvm.maxnum.f32",       "_Z4fmaxff"},
    {"fminf",                 "_Z4fminff"},
    {"llvm.minnum.f32",       "_Z4fminff"},
    {"fabsf",                 "_Z4fabsf"},
    {"llvm.fabs.f32",         "_Z4fabsf"},
    {"ceilf",                 "_Z4ceilf"},
    {"llvm.ceil.f32",         "_Z4ceilf"},
    {"floorf",                "_Z5floorf"},
   {"llvm.floor.f32",        "_Z5floorf"},

   // double:
   {"sin",                   "_Z3sind"},
   {"cos",                   "_Z3cosd"},
   {"tan",                   "_Z3tand"},
   {"erf",                   "_Z3erfd"},
   {"exp",                   "_Z3expd"},
   {"log",                   "_Z3logd"},
   {"log2",                  "_Z4log2d"},
   {"pow",                   "_Z3powdd"},
   {"sqrt",                  "_Z4sqrtd"},
   {"fmax",                  "_Z4fmaxdd"},
   {"llvm.maxnum.f64",       "_Z4fmaxdd"},
   {"fmin",                  "_Z4fmindd"},
   {"llvm.minnum.f64",       "_Z4fmindd"},
   {"fabs",                  "_Z4fabsd"},
   {"llvm.fabs.f64",         "_Z4fabsd"},
   {"ceil",                  "_Z4ceild"},
   {"llvm.ceil.f64",         "_Z4ceild"},
   {"floor",                 "_Z5floord"},
   {"llvm.floor.f64",        "_Z5floord"},
   {"invsqrtf",              "_Z5rsqrtf"},
   {"invsqrt",               "_Z5rsqrtd"}};
```

高速逆平方根関数

注: `libomptarget` ランタイム・ライブラリーには、GPU カーネルの開始/終了時間とデータ転送時間を追跡するパフォーマンス・プロファイル機能が実装されています。この機能を有効にするには、環境変数 `LIBOMPTARGET_PROFILE=T` を設定します。結果は次のようになります。

```
GPU Performance (Gen9, export LIBOMPTARGET_PROFILE=T,usec)
… …
Kernel Name:
__omp_offloading_811_29cbc383__ZN12BlackScholesIdE12execute_partEiii_l368
iteration #0 ...
calling validate ... ok
calling close ...
execution finished in 1134.914ms, total time 0.045min
passed
LIBOMPTARGET_PROFILE:
-- DATA-READ: 16585.256 usec
-- DATA-WRITE: 9980.499 usec
-- EXEC-
__omp_offloading_811_29cbc383__ZN12BlackScholesIfE12execute_partEiii_l368:
24048.503 usec
```

データの読み取りとそのコストは、OpenCL* SVMMap/SVMUnmap とデータコピーを測定しました。以下に例を示します。

```
INVOKE_CL_RET_FAIL(clEnqueueSVMMap, queue, CL_TRUE, CL_MAP_WRITE, tgt_ptr,
size, 0, nullptr, nullptr);
memcpy(tgt_ptr, hst_ptr, size);
INVOKE_CL_RET_FAIL(clEnqueueSVMUnmap, queue, tgt_ptr, 0, nullptr, nullptr);
```

サポートされる USM は、L0 API のサポート・スケジュールに合わせて OpenMP* 5.0 の USM 機能をサポートする予定です。

GPU 固有のデバッグ情報の事前統合

GPU 固有のデバッグを可能にするため、`export LIBOMPTARGET_DEBUG=1` 環境変数がサポートされました。これにより、オフロードのランタイムデバッグ情報を取得できます。デフォルト値は 0 であり、オフロードのランタイムデバッグ情報を出力しないことを意味します。以下の例をご覧ください。

```
sh-4.2$ export LIBOMPTARGET_DEBUG=1
sh-4.2$ ./matmul

Libomptarget --> Loading RTLs...
Libomptarget --> Loading library 'libomptarget.rtl.nios2.so'...
Libomptarget --> Unable to load library 'libomptarget.rtl.nios2.so':
libomptarget.rtl.nios2.so: cannot open shared object file: No such file or
directory!
Libomptarget --> Loading library 'libomptarget.rtl.x86_64_mic.so'...
Libomptarget --> Successfully loaded library 'libomptarget.rtl.x86_64_mic.so'!
Libomptarget --> No devices supported in this RTL
Libomptarget --> Loading library 'libomptarget.rtl.opencl.so'...
Target OPENCL RTL --> Start initializing OpenCL
Target OPENCL RTL --> cl platform version is OpenCL 2.1 LINUX
Target OPENCL RTL --> Found 1 OpenCL devices
Target OPENCL RTL --> Device#0: Genuine Intel(R) CPU 0000 @ 3.00GHz
Target OPENCL RTL --> max WGs is: 8
Target OPENCL RTL --> max WG size is: 8192
Target OPENCL RTL --> addressing mode is 64 bit
Target OPENCL RTL --> cl platform version is OpenCL 2.1
Target OPENCL RTL --> Found 1 OpenCL devices
Target OPENCL RTL --> Device#0: Intel(R) Gen9 HD Graphics NEO
```

```
Target OPENCL RTL --> max WGs is: 24
Target OPENCL RTL --> max WG size is: 256
Target OPENCL RTL --> addressing mode is 64 bit
Libomptarget --> Successfully loaded library 'libomptarget.rtl.opencl.so'!
Libomptarget --> Optional interface: __tgt_rtl_run_target_team_nd_region
Libomptarget --> Optional interface: __tgt_rtl_run_target_region_nowait
Libomptarget --> Optional interface: __tgt_rtl_run_target_team_region_nowait
Libomptarget --> Optional interface: __tgt_rtl_run_target_team_nd_region_nowait
Libomptarget --> Registering RTL libomptarget.rtl.opencl.so supporting 1 devices!
Libomptarget --> Loading library 'libomptarget.rtl.ppc64.so'...
Libomptarget --> Unable to load library 'libomptarget.rtl.ppc64.so':
libomptarget.rtl.ppc64.so: cannot open shared object file: No such file or
directory!
Libomptarget --> Loading library 'libomptarget.rtl.x86_64.so'...
Libomptarget --> Unable to load library 'libomptarget.rtl.x86_64.so':
libomptarget.rtl.x86_64.so: cannot open shared object file: No such file or
directory!
Libomptarget --> Loading library 'libomptarget.rtl.cuda.so'...
Libomptarget --> Unable to load library 'libomptarget.rtl.cuda.so':
libomptarget.rtl.cuda.so: cannot open shared object file: No such file or
directory!
Libomptarget --> Loading library 'libomptarget.rtl.aarch64.so'...
Libomptarget --> Unable to load library 'libomptarget.rtl.aarch64.so': libomptar
get.rtl.aarch64.so: cannot open shared object file: No such file or directory!
Libomptarget --> RTLs loaded!
Target OPENCL RTL --> Target binary is VALID
Libomptarget --> Image 0x000000000060d0a0 is compatible with RTL
libomptarget.rtl.opencl.so!
Libomptarget --> RTL 0x0000000002012250 has index 0!
Libomptarget --> Registering image 0x000000000060d0a0 with RTL
libomptarget.rtl.opencl.so!
Libomptarget --> Done registering entries!
Libomptarget --> Entering target region with entry point 0x000000000040a2a0
anddevice Id -1
Libomptarget --> Checking whether device 0 is ready.
Libomptarget --> Is the device 0 (local ID 0) initialized? 0
Target OPENCL RTL --> Initialize OpenCL device
Libomptarget --> Device 0 is ready to use.
Target OPENCL RTL --> Dev 0: load binary from 0x000000000060d0a0 image
Target OPENCL RTL --> Expecting to have 1 entries defined.
Target OPENCL RTL --> Found device RTL:
/home/users/xtian/cmplr/dev_xmain/builds/xmainoffloadlinuxefi2_debug/llvm/lib/lib
omptarget-opencl.a

** Program Scope patch lists **
Libomptarget --> Creating new map entry: HstBase=0x00000000006224f0, HstBegin=0x
00000000006224f0, HstEnd=0x000000000062c130, TgtBegin=0x0000000002100ad0
Libomptarget --> There are 40000 bytes allocated at target address 0x00000000021
00ad0 - is new
… … … …
Libomptarget --> Launching target execution __omp_offloading_811_600f59f6__Z7Com
putev_l14 with pointer 0x0000000002a994a0 (index=0).
Target OPENCL RTL --> OpenCL: Kernel Arg 0 set successfully
Target OPENCL RTL --> OpenCL: Kernel Arg 1 set successfully
Target OPENCL RTL --> OpenCL: Kernel Arg 2 set successfully
… … …
Libomptarget --> Image 0x000000000060d0a0 is compatible with RTL
0x0000000002012250!
Libomptarget --> Unregistered image 0x000000000060d0a0 from RTL
0x0000000002012250!
Libomptarget --> Done unregistering images!
Libomptarget --> Removing translation table for descriptor 0x000000000060ea50
Libomptarget --> Done unregistering library!
```

注: インテル® GPU を使用するプログラミングは、ほかの GPU 向けのプログラミングに似ています。異なる GPU (マイクロ) アーキテクチャーは異なる動作になります。新しい (マイクロ) アーキテクチャー向けにコードをチューニングするのは、機能の移行よりも困難です。インテルは、後者の負担を軽減するためコンパイラー、ライブラリー、およびツールの提供に取り組んでいますが、これによってパフォーマンス最適化の要求がなくなるわけではありません。

さらに詳しく

ドキュメント	説明とリンク
OpenMP* 5.0 仕様 (PDF)	OpenMP* 仕様 (英語) では OpenMP* オフロードをデバイス向けに使用する方法を説明しています。
GNU* C/C++ ライブラリー	2 つの ABI の使用 (英語) についての説明です。
OpenMP* のサポートに関する SC'16 と SC'17 LLVM-HPC ワークショップの文書	**明示的な並列化と SIMD ベクトル化のための LLVM コンパイラーの実装。** LLVM-HPC@SC 2017: 4:1-4:11 **並列化、SIMD ベクトル化、オフロードのための LLVM フレームワークと IR 拡張。** LLVM-HPC@SC 2016: 21-31

法務上の注意書きと最適化に関する注意事項

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。実際の性能はシステム構成によって異なります。

絶対的なセキュリティーを提供できるコンピューター・システムはありません。

本資料は、(明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず) いかなる知的財産権のライセンスも許諾するものではありません。

本資料で説明されている製品およびサービスには、不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

インテルは、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。

* 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。

製品とパフォーマンス情報

[1] インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804