Is OpenMP 4.5 Target Off-load Ready for Real Life?  
A Case Study of Three Benchmark Kernels

Jose M. Monsalve Diaz (UDEL), Gabriele Jost (NASA), Sunita Chandrasekaran (UDEL) and Sergio Pino(ex-UDEL)  
September 25, 2018  
Talk at OpenMPCon, 2018
Outline

• Introduction
  – The OpenMP Target Concept

• Benchmark implementations
  – Benchmark descriptions
  – Porting OpenACC to OpenMP 4.5

• Performance Analysis:
  – Comparing compilers
  – Comparing hardware
  – Comparing OpenMP 4.5 to OpenACC

• Summary and Conclusions
• Discussion
Introduction

Number of systems with Accelerator devices in the Top500 list

Programming models
- Device specific language
- Device specific libraries
- Directives
  - OpenMP (since 1998)
  - OpenACC (since 2011)
OpenMP and device offloading

#pragma omp…

<table>
<thead>
<tr>
<th>OpenMP 4.0</th>
<th>OpenMP 4.5</th>
<th>OpenMP 5.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>target [data]</td>
<td>taskloop</td>
<td>allocate</td>
</tr>
<tr>
<td>declare target</td>
<td>taskloop simd</td>
<td>declare mapper</td>
</tr>
<tr>
<td>target update</td>
<td>target enter data</td>
<td>Memory spaces</td>
</tr>
<tr>
<td>simd</td>
<td>target exit data</td>
<td>parallel loop</td>
</tr>
<tr>
<td>declare simd</td>
<td>target simd</td>
<td>teams loop</td>
</tr>
<tr>
<td>loop simd</td>
<td>… Other API Calls …</td>
<td>… Other API Calls …</td>
</tr>
<tr>
<td>parallel loop simd</td>
<td>… Other API Calls …</td>
<td>… Other API Calls …</td>
</tr>
<tr>
<td>teams</td>
<td>… and other API Calls …</td>
<td>… and other API Calls …</td>
</tr>
</tbody>
</table>

| distribute [simd] | distribute parallel for [simd] | teams distribute [simd] |
| distribute parallel for [simd] | teams distribute parallel for [simd] | target teams |
| teams distribute [simd] | target teams distribute parallel for [simd] | target teams distribute [simd] |
| … and other API Calls … | … and other API Calls … | … and other API Calls … |
Important OpenMP Constructs and Clauses

#pragma omp target or !$omp target
- Create data environment and execute code region on the device

#pragma omp target map(map-type: list)
- Map a variable to/from the device data environment

#pragma teams
- Start kernel on the GPUs

#pragma teams distribute, parallel for, simd
- Distribute the work across the teams and threads within each team
Laplace Kernel Example

```c
#pragma omp target teams distribute
for( int j = 1; j < n-1; j++ ) {
    #pragma parallel for reduction(max:error)
    for( int i = 1; i < m-1; i++ ) {
        error = fmax( error, fabs(Anew[j][i] - A[j][i]) );
    }
}
```

For more details check out the presentation by Jeff Larkin, Nvidia:

Image courtesy of Nvidia
NPB Benchmark Descriptions

- **FT** = Discrete 3D Fast Fourier Transform
  - Requires all-to-all data data transfers
  - Compiler Challenges:
    - Usage of complex data structures required manually handling real and imaginary parts separately; function calls in inner loops benefit from manual inline of function calls

- **LU-HP** = Lower-Upper Gauss Seidel Solver using a hyperplane method
  - A pipelined algorithm requires explicit thread-to-thread synchronization, which is not suitable for device execution
  - Compiler Challenges:
    - Data layout is not optimal for device execution; shared array data structures increase data transfer

- **MG** = Multi-Grid Solvers on a sequence of meshes
  - Requires long and short distance data transfers between grids
  - Memory intensive
  - Compiler Challenges:
    - 3D data structures required manual linearization

- NPB benchmark offers different classes (Problem size) – S thru E
Development methodology

Initial steps

Previous work on NPB OpenACC Benchmark

LU

MG

FT

Workload distribution among developers

OpenMP offloading development cycle

Select offloading sections

Detect possible pitfalls

Identify data movements

Create target regions

No parallelism

Check correctness

Obtain base performance

Apply constructs for parallelism

Find optimization opportunities

Obtain performance metrics
General Implementation Strategy: Translating OpenACC to OpenMP

- Start out with the existing NPB 2.5 OpenACC Implementation developed in 2014 by Xu et al. (see Ref 1.)
- Translate OpenACC to OpenMP 4.5 matching constructs if available

<table>
<thead>
<tr>
<th>Open ACC 2.5</th>
<th>OpenMP 4.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>target</td>
</tr>
<tr>
<td>{enter, exit} data</td>
<td>target {enter, exit} data</td>
</tr>
<tr>
<td>parallel present (a1,a2,...)</td>
<td>target map (alloc: a1, a2,...)</td>
</tr>
<tr>
<td>loop [gang/worker/vector]</td>
<td>distribute / parallel for /simd</td>
</tr>
<tr>
<td>device_ptr</td>
<td>is_device_pointer</td>
</tr>
<tr>
<td>kernels</td>
<td>----</td>
</tr>
</tbody>
</table>

Tell compiler that data is already present on the device

Not available in OpenMP!
Places burden of dependence analysis on the compiler

Compiler will detect if data is already present on the device
3D partial differential equation using an Fast Fourier Transform (FFT)

- Complex data:
  - Treat real and imaginary parts separately as in OpenACC
- Many function calls in inner loops
  - Manually inline function calls as in OpenACC

```c
#pragma acc parallel num_gangs(d3) vector_length(128) \ 
         present(gty1_real,gty1_imag,gty2_real,gty2_imag,\ 
                   u1_real,u1_imag,u_real,u_imag)\ 
#pragma omp target map ( alloc: u1_real, u1_imag, u_real, u_imag)\ 
        map(from: gty1_real, gty1_imag, gty2_real, gty2_imag)\ 
{
  #pragma acc loop gang independent
  #pragma omp teams distribute collapse(2)
  for (k = 0; k < d3; k++) {}\ 
  #pragma acc loop vector independent
  for (l = 1; l <= logd1; l += 2){\ 
    #pragma omp parallel for collapse(2) private(i11, i12, i21, i22, uu1_real, uu1_imag,\ 
                        x11_real, x11_imag, x21_real, x21_imag, temp_real, temp_imag)\ 
    for (i1 = 0; i1 <= li - 1; i1++) {\ 
      for (k1 = 0; k1 <= lk - 1; k1++) {\ 
        ...
      }\ 
  }\ 
  ...
  gty2_real[k][i21+k1][j] = x11_real + x21_real;
  ...
  temp_real = x11_real - x21_real;
  gty2_real[k][i22+k1][j] = (uu1_real)*(temp_real) - (uu1_imag)*(temp_imag);
```
LU-HP Implementation

Lower-Upper Gauss Seidel Solver using a hyperplane method

- **Compiler Challenges:**
  - Array privatization
  - Change data layout to enable memory coalescing
  - Manual loop unrolling

```c
#include <stdio.h>
#include <stdlib.h>

#define RADIUS 1.0
#define DT 0.01
#define NPL 100
#define M 100
#define N 100
#define L 100
#define K 100
#define P 100
#define Q 100

int main()
{
    float *a, *b, *c, *d, *u;
    int *indxp, *jndxp;
    double rho_i[NPL+1], qs[NPL+1];

    // Initialize and allocate memory
    ...

    // Solver loop
    for (n = 1; n <= npl; n++) {
        j = jndxp[l][n];
        i = indxp[l][n];
        k = l - i - j;
        tmp1 = rho_i[k][j][i];
        tmp2 = tmp1 * tmp1;
        d[0][0][n] = 1.0 + dt * 2.0 * ( tx1 * dx1 + ty1 * dy1 + tz1 * dz1 );
        d[0][1][n] = -dt * 2.0 * ( tx1 * r43 + ty1 + tz1 ) * c34 * tmp2 * u[1][k][j][i];
        d[1][1][n] = 1.0 + dt * 2.0 * c34 * tmp1 * ( tx1 * r43 + ty1 + tz1 ) + dt * 2.0 * ( tx1 * dx2 + ty1 * dy2 + tz1 * dz2 );
        ...
    }

    // Free memory
    ...

    return 0;
}
```
Multi-Grid Solvers on a sequence of meshes

- Long and short distance data transfers between grids; memory bandwidth intensive
- Compiler Challenges:
  - 3D data structures required manual linearization

```c
#define I3D(array, n1, n2, i3, i2, i1) (array[(i3)*n2*n1 + (i2)*n1 + (i1)])
```

```c
r1 = (double*)acc_malloc(n3*n2*n1*sizeof(double))
r1 = (double*)omp_target_alloc(n3*n2*n1*sizeof(double), omp_get_default_device());
```

```c
#pragma acc data deviceptr(u1,u2), present(ou[0:n3*n2*n1]),
    present(ov[0:n3*n2*n1], or[0:n3*n2*n1])nt n3)
#pragma acc parallel num_gangs(n3-2) num_workers(8) vector_length(128)
#pragma acc omp target map(tofrom: ou[0:n3*n2*n1]) map(tofrom: ov[0:n3*n2*n1])
    map(tofrom: or[0:n3*n2*n1]) is_device_ptr(u1, u2)
#pragma acc loop gang independent
#pragma acc omp teams distribute
    for (i3 = 1; i3 < n3-1; i3++) {
        #pragma acc loop worker independent
        #pragma acc omp parallel for collapse(2)
            for (i2 = 1; i2 < n2-1; i2++) {
                #pragma acc loop vector independent
                    for (i1 = 0; i1 < n1; i1++) {
                        I3D(u1, n1, n2, i3, i2, i1) = I3D(ou, n1, n2, i3, i2-1, i1)
                            + I3D(ou, n1, n2, i3, i2+1, i1)
                            + I3D(ou, n1, n2, i3-1, i2, i1)
                            + I3D(ou, n1, n2, i3+1, i2, i1);
                    }
                }
            }
        }
    }
```
### Evaluation Environment

**Challenge for our study:**

- **Different set of compilers available on different platforms**
  - Each behaving differently (correctness is not always portable)

- **What do we compare?**
  - Selection of systems with overlapping compilers
    - Support for OpenMP Offloading in HPC system is still low despite compilers support
    - Clang trunk bug with `math.h` and host specific `asm` code

<table>
<thead>
<tr>
<th></th>
<th><strong>Titan</strong></th>
<th><strong>Summit</strong></th>
<th><strong>Summitdev</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Cray XK7</td>
<td>IBM AC922</td>
<td>IBM S822LC</td>
</tr>
<tr>
<td>Nodes</td>
<td>6274</td>
<td>9216</td>
<td>54</td>
</tr>
<tr>
<td>CPU</td>
<td>16 cores AMD Opteron 6274</td>
<td>22 Cores IBM POWER9</td>
<td>20 cores IBM POWER8</td>
</tr>
<tr>
<td>Accelerators</td>
<td>1 NVIDIA K20X</td>
<td>4 NVIDIA P100</td>
<td>6 NVIDIA V100</td>
</tr>
</tbody>
</table>
Compilers

<table>
<thead>
<tr>
<th></th>
<th>Titan</th>
<th>Summit</th>
<th>Summitdev</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCC</td>
<td>-</td>
<td>-</td>
<td>7.1.1</td>
</tr>
<tr>
<td>PGI</td>
<td>18.5</td>
<td>18.3</td>
<td>18.4</td>
</tr>
<tr>
<td>CCE</td>
<td>8.7.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CLANG/LLVM</td>
<td>-</td>
<td>CORAL 3.8.0</td>
<td>CORAL 3.8.0</td>
</tr>
<tr>
<td>XLC</td>
<td>-</td>
<td>16.1.0</td>
<td>13.1.0</td>
</tr>
</tbody>
</table>

Challenge for our study:

- Different set of compilers available on different platforms
  - Each behaving differently (correctness is not always portable)

- What do we compare?
  - Selection of systems with overlapping compilers
    - Support for OpenMP Offloading in HPC system is still low despite compilers support
    - Clang trunk bug with math.h and host specific asm code
Run time on Summitdev

- Observations:
  - Runtimes of XL and Clang is quite similar
  - xlc V13 failed verification for FT
  - GCC 7.1.1 low performance
  - PGI-OpenACC 18.1 shows relatively better performance
  - PGI supports OpenMP 4.5 in their LLVM compiler, but there is no offload support yet
  - Class C FT and LU-HP fail due to memory
• Summit is currently unavailable due to acceptance effort …
• PGI does not support OMP Offloading yet
• Original OpenACC employs "pragma acc kernels" which is not available in OpenMP 4.5
Run time on Titan

• Observations:
  - For LU-HP and FT OpenACC significantly outperforms OpenMP 4.5
  - Only for MG OpenMP 4.5 can keep up with OpenACC
Comparing Compilers: MG Class A xlc and gcc on Summitdev

**xlc v 13**

```
==67718== Profiling result:

<table>
<thead>
<tr>
<th>Type</th>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU activities:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.34%</td>
<td>292.94ms</td>
<td>810</td>
<td>361.65us</td>
<td>4.4800us</td>
<td>1.3336ms</td>
<td>__xl_resid_l679_OL_4</td>
<td></td>
</tr>
<tr>
<td>20.51%</td>
<td>237.16ms</td>
<td>810</td>
<td>292.79us</td>
<td>5.2800us</td>
<td>1.1059ms</td>
<td>__xl_resid_l672_OL_3</td>
<td></td>
</tr>
<tr>
<td>14.04%</td>
<td>162.31ms</td>
<td>808</td>
<td>200.88us</td>
<td>3.1680us</td>
<td>1.3819ms</td>
<td>__xl_psinv_l551_OL_2</td>
<td></td>
</tr>
<tr>
<td>11.04%</td>
<td>127.61ms</td>
<td>808</td>
<td>157.93us</td>
<td>3.9040us</td>
<td>1.1059ms</td>
<td>__xl_psinv_l550_OL_1</td>
<td></td>
</tr>
<tr>
<td>0.33%</td>
<td>12.128us</td>
<td>14</td>
<td>866ns</td>
<td>704ns</td>
<td>1.6000us</td>
<td>[CUDA memcpy HtoD]</td>
<td></td>
</tr>
</tbody>
</table>
```

**gcc 7.1**

```
==45872== Profiling result:

<table>
<thead>
<tr>
<th>Type</th>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU activities:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34.59%</td>
<td>13.3874s</td>
<td>810</td>
<td>16.528ms</td>
<td>532.87us</td>
<td>62.394ms</td>
<td>resid$_omp_fn$44</td>
<td></td>
</tr>
<tr>
<td>18.63%</td>
<td>7.21199s</td>
<td>808</td>
<td>8.9257us</td>
<td>158.34us</td>
<td>60.259ms</td>
<td>psinv$_omp_fn$40</td>
<td></td>
</tr>
<tr>
<td>11.81%</td>
<td>4.56960s</td>
<td>810</td>
<td>5.6415ms</td>
<td>384.61us</td>
<td>20.359ms</td>
<td>resid$_omp_fn$46</td>
<td></td>
</tr>
<tr>
<td>7.07%</td>
<td>2.73749s</td>
<td>808</td>
<td>3.3880ms</td>
<td>129.28us</td>
<td>20.859ms</td>
<td>psinv$_omp_fn$42</td>
<td></td>
</tr>
<tr>
<td>3.30%</td>
<td>1.27857s</td>
<td>707</td>
<td>1.8084ms</td>
<td>123.84us</td>
<td>8.7265ms</td>
<td>rprj3$_omp_fn$16</td>
<td></td>
</tr>
<tr>
<td>0.00%</td>
<td>12.128us</td>
<td>14</td>
<td>866ns</td>
<td>704ns</td>
<td>1.6000us</td>
<td>[CUDA memcpy HtoD]</td>
<td></td>
</tr>
</tbody>
</table>
```

**Observations:**

- gcc shows a larger number of host-to-device data transfer
- xlc uses just a small number of asynchronous data transfers (cuMemcpyHtoDAsync)
- gcc and xlc employ different grid size, block size and number of registers per thread (see following 2 slides)
GCC: Memory copies to device interleaved with kernel execution

xlc: No memory copies
### Comparing OpenMP 4.5 vs OpenACC

**Performance FT on Titan**

#### OpenMP 4.5 + Cray cc

<table>
<thead>
<tr>
<th>Type</th>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU activities:</td>
<td>24.12%</td>
<td>2.20986s</td>
<td>6</td>
<td>368.31ms</td>
<td>368.29ms</td>
<td>368.36ms</td>
<td>cffts1_neg</td>
</tr>
<tr>
<td></td>
<td>7.94%</td>
<td>727.67ms</td>
<td>58</td>
<td>12.546ms</td>
<td>1.3440us</td>
<td>131.51ms</td>
<td>[CUDA memcpy DtOH]</td>
</tr>
<tr>
<td></td>
<td>3.40%</td>
<td>311.86ms</td>
<td>56</td>
<td>5.5689ms</td>
<td>928ns</td>
<td>41.782ms</td>
<td>[CUDA memcpy HtD]</td>
</tr>
<tr>
<td></td>
<td>7.91%</td>
<td>45.793ms</td>
<td>6</td>
<td>7.6321ms</td>
<td>7.5692ms</td>
<td>7.7115ms</td>
<td>cffts1_neg</td>
</tr>
<tr>
<td></td>
<td>5.36%</td>
<td>30.988ms</td>
<td>6</td>
<td>5.1646ms</td>
<td>5.0361ms</td>
<td>5.3973ms</td>
<td>cffts1_neg</td>
</tr>
</tbody>
</table>

#### OpenACC + Cray cc

<table>
<thead>
<tr>
<th>Type</th>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU activities:</td>
<td>32.83%</td>
<td>258.24ms</td>
<td>6</td>
<td>43.040ms</td>
<td>42.819ms</td>
<td>43.168ms</td>
<td>cffts1_neg</td>
</tr>
<tr>
<td></td>
<td>13.09%</td>
<td>102.99ms</td>
<td>6</td>
<td>17.165ms</td>
<td>928ns</td>
<td>25.769ms</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td></td>
<td>0.00%</td>
<td>9.9200us</td>
<td>6</td>
<td>1.6530us</td>
<td>1.4720us</td>
<td>1.9200us</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
</tbody>
</table>

#### Observations:
- For each loop there are 3 kernels for OpenMP 4.5 vs 1 kernel for OpenACC
- Data transfer to device is greatly reduced for OpenACC
- Data transfer to host is very high in OpenMP 4.5:
  - We had to move some arrays back to host to ensure correct execution, not necessary for OpenACC
Comparing Hardware: Performance Summitdev vs Summit using clang 3.8

• Observations:
  - We use the same compiler version on both platforms
  - We compare impact of using 1 Nvidia P100 GPU (Summitdev) vs 1 Nvidia V100 GPU (Summit)

Compiler bug with PPC: relocation truncated to fit: R_PPC64_TOC16_HA against `.bss`+
additional relocation overflows omitted from the output
Invitation to my next talk
IWOMP Thursday 2:30pm session

OpenMP 4.5 Validation and verification Suite for Device offload

Jose Monsalve Diaz        Swaroop Pophale
Oscar Hernandez  David E. Bernholdt Sunita Chandrasekaran
Summary

• We described our experiences porting 3 NPB benchmarks to OpenMP 4.5 w/ offloading
• We tested our implementations on 3 different systems at OCLF
• We compared compilers, programming models and hardware
• Conclusions:
  - Evaluating 3 NPB benchmarks showed that 4.5 target offload did not lack a feature/functionality when compared with OpenACC
  - OpenMP 4.5 employs existing functionality for accelerator execution, if possible, e. g. “parallel for”, and “simd”
  - Compiler support for OpenMP would definitely benefit from further improvement
• User community:
  - Would you find it useful to have a public domain, full NPB OpenMP target implementation available?
References


Images