OpenMP Extensions for Heterogeneous Architectures

Leo White

University of Cambridge

7th International Workshop on OpenMP
Heterogeneity in Modern Architectures

Heterogeneous processors

- Accelerators
- General-Purpose computing on Graphical Processing Units (GPGPU)

Heterogeneous memory systems

- Non-Uniform Memory Access (NUMA)
- Partitioned address spaces
Problems for OpenMP

There are two orthogonal problems for OpenMP with heterogeneous architectures:

1. OpenMP assumes a single shared address space.
2. There is no mechanism in OpenMP for allocating work to specific processors on an architecture.

This talk discusses the second problem.
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma omp parallel
    {
        #pragma omp for nowait
        for (i=1; i<n; i++)
            b[i] = (a[i] + a[i-1]) / 2.0;
        #pragma omp for nowait
        for (i=0; i<m; i++)
            y[i] = sqrt(z[i]);
    }
}
```
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;

    #pragma omp parallel
    {
        #pragma omp for nowait
        for (i=1; i<n; i++)
        
            b[i] = (a[i] + a[i-1]) / 2.0;

        #pragma omp for nowait
        for (i=0; i<m; i++)
            y[i] = sqrt(z[i]);
    }
}```
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma omp parallel
    {
        #pragma omp for nowait
        for (i=1; i<n; i++)
            b[i] = (a[i] + a[i-1]) / 2.0;
        #pragma omp for nowait
        for (i=0; i<m; i++)
            y[i] = sqrt(z[i]);
    }
}
```
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma parallel omp for
    for (i=1; i<n; i++)
        b[i] = (a[i] + a[i-1]) / 2.0;
    #pragma parallel omp for
    for (i=0; i<m; i++)
        y[i] = sqrt(z[i]);
}
```
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma omp parallel
    {
        if(get_thread_num() == 0) {
            #pragma parallel omp for
            for (i=1; i<n; i++)
                b[i] = (a[i] + a[i-1]) / 2.0;
        } else {
            #pragma parallel omp for
            for (i=0; i<m; i++)
                y[i] = sqrt(z[i]);
        }
    }
}
```
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma omp parallel
    {
        #pragma omp for nowait
        for (i=1; i<n; i++)
            b[i] = (a[i] + a[i-1]) / 2.0;
        #pragma omp for nowait
        for (i=0; i<m; i++)
            y[i] = sqrt(z[i]);
    }
}
Example 2

```c
void process_list_items(node *head)
{
    #pragma omp parallel
    {
        #pragma omp single
        {
            node *p = head;
            while (p) {
                #pragma omp task
                process(p);
                p = p->next;
            }
        }
    }
}
```

[Diagram: Main Processor = blue; Accelerators = red]
void process_list_items(node *head)
{
    #pragma omp parallel
    {
        #pragma omp single
        {
            node *p = head;
            while (p) {
                #pragma omp task
                process(p);
                p = p->next;
            }
        }
    }
}
#pragma omp parallel subteams($name_1($procs_1)[size_1], \ldots)$

Each argument represents a subteam:

- $size_n$ is the number of threads in the subteam.
- $name_n$ is the *subteam name* that refers to the subteam within this parallel region.
- $procs_n$ is an expression that represents the processors to which the subteam is mapped.
Processor Expressions

These are implementation-defined expressions with new type `omp_procs_t`, whose values represent sets of processors in an architecture. For example:

- PROC_A
- (PROC_A | PROC_C)
- omp_get_proc(2)

Multiple values can represent the same set of processors e.g.:

- ACCELERATORS
- SCATTER(ACCELERATORS)

Both map threads to the accelerators, but the second one keeps the threads as far apart as possible.
On Clause

\[ \text{on}(\text{subteams}_1, \text{subteams}_2, \ldots) \]

Each argument is a subteam name.

Can be used:

- with workshare constructs
- with the \texttt{task} construct
- as a directive, similar to the \texttt{master} construct
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z) {
    int i;
    #pragma omp parallel subteams(master[1], others(PROC2)[4])
    {
        #pragma omp on(others)
        {
            #pragma omp for nowait
            for (i=1; i<n; i++)
                b[i] = (a[i] + a[i-1]) / 2.0;
            #pragma omp for nowait
            for (i=0; i<m; i++)
                y[i] = sqrt(z[i]);
        }
    }
}
```
Example 1

```c
void a9(int n, int m, float *a, float *b, float *y, float *z)
{
    int i;
    #pragma omp parallel subteams(st1(PROC1)[4], st2(PROC2)[4])
    {
        #pragma omp for nowait on(st1)
        for (i=1; i<n; i++)
            b[i] = (a[i] + a[i-1]) / 2.0;
        #pragma omp for nowait on(st2)
        for (i=0; i<m; i++)
            y[i] = sqrt(z[i]);
    }
}
```
Example 2

```c
void process_list_items(node *head)
{
    #pragma omp parallel subteams(master[1], accs(ACC)[5])
    {
        #pragma omp single on(master)
        {
            node *p = head;
            while (p) {
                #pragma omp task on(accs)
                process(p);
                p = p->next;
            }
        }
    }
}
```
Portability/Adaptability

How can these extensions be used write portable or adaptable programs?

- How can we write programs for multiple similar architectures?
- What about architectures that have variable resources?
- E.g. An appropriate GPU may or may not be available at runtime.

Integrate the extensions with the upcoming error model:

- Emit an error if the thread mapping is not possible.
- Allows program to try the best allocation and then fall back on an alternative.
- E.g. Try to use GPGPU but fall back to CPU if an appropriate GPU is not available.
Prototype Implementation

Cell Broadband Engine

- 1 PowerPC Processing Element (PPE)
- 7 Synergistic Processing Elements (SPE) with private local memories

Benchmarks

EP  An embarrassingly parallel algorithm with very little communication between threads.

IS  An integer sort with regular accesses to a shared array.

CG  A matrix-based algorithm with irregular accesses to shared arrays.
## Results

### EP

<table>
<thead>
<tr>
<th>PPE Threads</th>
<th>SPE Threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.23</td>
<td>2.45</td>
<td>3.68</td>
<td>4.90</td>
<td>6.12</td>
<td>7.35</td>
<td>7.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2.02</td>
<td>3.01</td>
<td>4.04</td>
<td>4.98</td>
<td>6.03</td>
<td>7.01</td>
<td>7.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.68</td>
<td>2.51</td>
<td>3.34</td>
<td>4.17</td>
<td>5.01</td>
<td>5.85</td>
<td>6.68</td>
<td>7.43</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.62</td>
<td>2.17</td>
<td>2.71</td>
<td>3.24</td>
<td>3.76</td>
<td>4.32</td>
<td>4.80</td>
<td>5.31</td>
<td></td>
</tr>
</tbody>
</table>

### IS

<table>
<thead>
<tr>
<th>PPE Threads</th>
<th>SPE Threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.07</td>
<td>0.14</td>
<td>0.20</td>
<td>0.27</td>
<td>0.33</td>
<td>0.39</td>
<td>0.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.14</td>
<td>0.20</td>
<td>0.27</td>
<td>0.33</td>
<td>0.39</td>
<td>0.44</td>
<td>0.42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.42</td>
<td>0.20</td>
<td>0.27</td>
<td>0.33</td>
<td>0.39</td>
<td>0.45</td>
<td>0.50</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1.18</td>
<td>0.27</td>
<td>0.33</td>
<td>0.39</td>
<td>0.45</td>
<td>0.50</td>
<td>0.55</td>
<td>0.49</td>
<td></td>
</tr>
</tbody>
</table>

### CG

<table>
<thead>
<tr>
<th>PPE Threads</th>
<th>SPE Threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</td>
<td>0.20</td>
<td>0.30</td>
<td>0.40</td>
<td>0.50</td>
<td>0.6</td>
<td>0.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.21</td>
<td>0.31</td>
<td>0.41</td>
<td>0.51</td>
<td>0.62</td>
<td>0.72</td>
<td>0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.64</td>
<td>0.31</td>
<td>0.41</td>
<td>0.51</td>
<td>0.62</td>
<td>0.72</td>
<td>0.82</td>
<td>0.21</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.5</td>
<td>0.29</td>
<td>0.33</td>
<td>0.37</td>
<td>0.40</td>
<td>0.41</td>
<td>0.43</td>
<td>0.15</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- There are two orthogonal problems with heterogeneous architectures for OpenMP:
  1. OpenMP assumes a single shared address space.
  2. There is no mechanism to allocate work to specific processors.

- The second problem can be solved by extending OpenMP with thread mapping and subteams.

- These extensions can be made more adaptable by integrating them with the future error model of OpenMP.