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

```
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]);
  }
}
```

■ = Main Processor ■ = Other Processor ■ = Both Processors

Example 1

```
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]);
  }
}
```

■ = Main Processor ■ = Other Processor ■ = Both Processors

Example 1

```
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]);
  }
}
```

■ = Main Processor ■ = Other Processor ■ = Both Processors

Example 1

```
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]);
}
```

■ = Main Processor ■ = Other Processor ■ = Both Processors

Example 1

```
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]);
      }
  }
}
```

■ = Main Processor ■ = Other Processor ■ = Both Processors

Example 1

```
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]);
  }
}
```

■ = Main Processor ■ = Other Processor ■ = Both Processors

Example 2

```
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;
          }
      }
  }
}
```

■ = Main Processor ■ = Accelerators

Example 2

```
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;
          }
      }
  }
}
```

■ = Main Processor ■ = Accelerators

Subteams Clause

```
#pragma omp parallel subteams(name1(procs1)[size1], ...)
```

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

`on(`$subteams_1$`, `$subteams_2$`, ...)`

Each argument is a subteam name.

Can be used:

- with workshare constructs
- with the `task` construct
- as a directive, similar to the `master` construct

Example 1

```
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

```
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

```
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

SPE Threads

PPE Threads	0	1	2	3	4	5	6	7
0	-	1.23	2.45	3.68	4.90	6.12	7.35	7.31
1	1	2.02	3.01	4.04	4.98	6.03	7.01	**7.98**
2	1.68	2.51	3.34	4.17	5.01	5.85	6.68	7.43
3	1.62	2.17	2.71	3.24	3.76	4.32	4.80	5.31

IS

SPE Threads

PPE Threads	0	1	2	3	4	5	6	7
0	-	0.07	0.14	0.20	0.27	0.33	0.39	0.36
1	1	0.14	0.20	0.27	0.33	0.39	0.44	0.42
2	**1.42**	0.20	0.27	0.33	0.39	0.45	0.50	0.45
3	1.18	0.27	0.33	0.39	0.45	0.50	0.55	0.49

CG

SPE Threads

PPE Threads	0	1	2	3	4	5	6	7
0	-	0.10	0.20	0.30	0.40	0.50	0.6	0.22
1	1	0.21	0.31	0.41	0.51	0.62	0.72	0.19
2	**1.64**	0.31	0.41	0.51	0.62	0.72	0.82	0.21
3	0.5	0.29	0.33	0.37	0.40	0.41	0.43	0.15

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.