OpenMP

Markus Höhnerbach and Prof. Paolo Bientinesi

HPAC, RWTH Aachen
hoehnerbach@aices.rwth-aachen.de

WS16/17

Tasks

- To date:
  - Data parallelism: `#pragma omp for`
  - Limited task parallelism: `#pragma omp sections`
- We may need more flexibility
- Can you think of such problems/algorithms?

Linked lists

```
#pragma omp parallel
{
    #pragma omp for
    for ( list_iter = &start;
          list_iter != NULL;
          list_iter = list_iter->next )
    {
        process( list_iter ); // e.g., testPrime( list_iter->value );
    }
}
```

Trees

```
void traverse( node *n )
{
    if ( n == NULL ) {
        return;
    traverse( n->left_child );
    traverse( n->right_child );
    process( n );
}
```

Code along

- `01a.linked-list.c`

How does it work?

- Thread encounters a task (`#pragma omp task`)
- Code in the structured block is packed (e.g., as a function)
- Data environment is initialized
- Task may be
  - immediately executed by the encountering thread or
  - defered to a later execution (pool of tasks)
- If defered, thread executing may be a different one

Data-sharing attributes

- More complicated than in parallel regions
- Some common rules:
  - Static and global variables are shared
  - Automatic (local) variables are private
- Additional rules:
  - If shared in the enclosing context: shared
  - Otherwise: firstprivate
  - Orphaned tasks: firstprivate

Data-sharing attributes

Example (source: Terboven & Schmidl)

```
int a = 1;
int main( void )
{
    int b = 2, c = 3;
    #pragma omp parallel
    #pragma omp parallel private (b)
    {
        int d = 4;
        #pragma omp task
        {
            int e = 5;

            // scope of a:   shared
            // scope of b:   firstprivate
            // scope of c:   shared
            // scope of d:   firstprivate
            // scope of e:   private
        }
    }
}
```

Data-sharing attributes

Example

```
int a = 1;
int function( int b )
{
    int c;
    #pragma omp task
    {
        int d = 5;

        // scope of a:    shared
        // scope of b:    firstprivate
        // scope of c:    firstprivate
        // scope of d:    private
    }
}
int main( void )
{
    int b = 1;
    #pragma omp parallel
    function( b );
}
```

Code along

- `02a.fibonacci-rec.c`

Synchronization

- `#pragma omp taskwait`: wait until child tasks complete execution
- `#pragma omp taskgroup`: wait until all descendant tasks complete execution
- `#pragma omp barrier`: wait until all tasks created by the threads in the team complete execution

Clauses

- `#pragma omp task [clause [, clause] ...]`
- The following clauses apply:
  - `private, firstprivate, shared, default`
  - `depend`
  - `priority`

Dependencies

- OpenMP allows us to indicate data dependencies among tasks:

  `depend(`*`dependency-type`*`:`*`list`*`)`

- where *dependency-type* is one of `in`, `out`, `inout`
- *list* is a list of variables or array sections
- Meaning of *dependency-type*:
  - `in`: the task depends on all previous sibling tasks which reference at least one of the list items in an `out` or `inout` dependency
  - `out, inout`: the task depends on all previous sibling tasks which reference at least one of the list items in an `in`, `out` or `inout` dependency

Dependencies

```
#pragma omp parallel
#pragma omp single
for( i = 2; i < N; i++ )
    #pragma omp task depend(in:fib[i-1],fib[i-2]) \
                     depend(out:fib[i]) \
                     firstprivate(i)
    fib[i] = fib[i-1] + fib[i-2];
```

- See `03a.fibonacci-iter.c` vs `03b.fibonacci-iter-reversed-loop.c`

Dependencies

```
#pragma omp parallel
#pragma omp single
for( i = 2; i < N; i++ )
    #pragma omp task depend(in:fib[i-1],fib[i-2]) \
                     depend(out:fib[i]) \
                     firstprivate(i)
    fib[i] = fib[i-1] + fib[i-2];
```

- See `03a.fibonacci-iter.c` vs
  `03b.fibonacci-iter-reversed-loop.c`
- Create the dependencies in the sequential order :)

Algorithm: A := CHOL(A)

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $\texttt{size}(A_T) < \texttt{size}(A)$ **do**

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} := \text{CHOL}(A_{11})$ (CHOL)

$A_{21} := A_{21} A_{11}^{-T}$ (TRSM)

$A_{22} := A_{22} - A_{21} A_{21}^{T}$ (SYRK)

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

Priorities

```
priority(priority-value)
```

- Hint to the compiler for the priority of the task
- Non-negative value
- The higher the value the higher the priority
- Lowest, default priority: 0
- To allow for higher values, set the environment variable `OMP_MAX_TASK_PRIORITY`