Reduce and Scan Patterns

A Short Introduction

Joseph Kehoe[1]

[1]Department of Computing and Networking
Institute of Technology Carlow

CDD101, 2017

Table of Contents

Table of Contents

Definition

- The Reduce pattern is where a sequence of input data is reduced down to a single output value.
- A Combiner function is applied to every member of the input set.
- The combiner function operates a a pair of input values **result=combine(a,b)**
- Examples include using $+$ to get the sum of a sequence
- We assume that the associated pairs of input can be combined in different orders and still get the same answers
- e.g. a+b+c = b+a+c = b+c+a = a+c+b = c+a+b = c+b+a

Simple Example

```
float sum(int dim,float in[])
{
        float sum=0.0;
        for (int i=0; i < dim; ++i)
        {
                sum +=in[i];
        }
        return sum;
}
```

Simple OpenMP Implementation

Reduction is built in for simple operators

```
float sum(int dim,float in[])
{
float sum=0.0;
#pragma omp simd parallel for reduction(+:sum)
for (int i=0; i < dim; ++i)
{
sum +=in[i];
}
return sum;
}
```

Table of Contents

More Complex Operators

- What if our operator is not supported by the **reduction** clause?
- Produce our own code using Tiling
- First each thread produces a result for its own subsequence
- Then combine all results for each tile into one value

Simple OpenMP Implementation

```
float sum(int dim, float in[])
{
float result=0.0;
float tileResult[NumThreads];
#pragma omp parallel for
for (int i=0; i < dim; ++i)
{
  int tid = omp_get_thread_num();
  tileResult[tid] =op(tileResult[tid],in[i]);
}
for (int i=0; i < NumThreads; ++i)
{
  result=op(tileResult[i],result);
}
return result;
}
```

Table of Contents

Definition

- Produces all partial reductions of an input sequence to produce a new output sequence
- used in e.g. integration
- Using sum as an example - each element will contain the sun of all previous elements
  - **Inclusive** scan means the sum includes everything up to and including the current element
  - **Exclusive** scan means we sum only the previous elements (do not include ourselves)
  - Show output of each type on sequence: 3, 4, 6, 8, 1, 4

Approach

If using standard fork join we use a three phase approach

1. Tile the sequence and reduce each tile in parallel
2. Do an eclusive scan of the reduction values (always exclusive)
3. Scan each of the tiles where the initial value is that calculated by the previous step

Or use **tasks** to implement a tree based approach