Introduction to FastFlow programming

SPM lecture, November 2015

Massimo Torquati <torquati@di.unipi.it>

Computer Science Department, University of Pisa - Italy

ClassWork4: comments

- Let's comment on a possible solution of the first version. Take a look in the ClassWork4 folder:

  ~spm1501/public/ClassWork4/primes_master-worker.cpp

- Please compare your solution with the second version provided in the same folder:

  ~spm1501/public/ClassWork4/primes_master-worker2.cpp

Data Parallel Computations

- In data parallel computations, large data structures are partitioned among the number of concurrent resources each one computing the same function (F) on the assigned partition
- Input data may come from an input stream
- Typically the function F may be computed independently on each partition
  - There can be dependencies as in stencil computations
- **Goal**: reduce the *completion time* for computing the input task
- Patterns:
  - map, reduce, stencil, scan,… typically they are encountered in sequential program as *loop-based computations*
- In FastFlow we decided to implement a sort of building-block for data-parallel computations that are the **ParallelFor/ParallelForReduce**

FastFlow ParallelFor

- The ParallelFor patterns can be used to parallelize loops with independent iterations
- The class interface is defined in the file *parallel_for.hpp*
- Example:

```
// A and B are 2 arrays of size N

for(long i=0; i<N; ++i)
    A[i] = A[i] + B[i];
```

```
#include <ff/parallel_for.hpp>
using namespace ff;

ParallelFor pf;   // defining the object

pf.parallel_for(0, N, 1, [&A,B](const long i) {
    A[i] = A[i] + B[i];
});
```

- Constructor interface (all parameters have a default value):
  - ***ParallelFor(maxnworkers, spinWait, spinBarrier)***
- parallel_for interface (on the base of the number and type of bodyF arguments you have different parallel_for methods):
  - ***parallel_for(first, last, step, chunk, bodyF, nworkers)***
  - ***bodyF is a C++ lambda-function***

FastFlow ParallelForReduce

- The ParallelForReduce patterns can be used to parallelize loops with independent iterations having reduction variables (map+reduce)
- Example:

```
// A is an array of long of size N
long sum = 0;
for(long i=0; i<N; ++i)
     sum += A[i];
```

```
#include <ff/parallel_for.hpp>
using namespace ff;

ParallelForReduce<long> pfr;
long sum=0;
pfr.parallel_reduce(sum, 0,
                    0,N,1, [](const long i, long &mysum) {
                mysum += A[i] + B[i];
    },
    [ ](long &s, const long e) { s += e;}
);
```

- The constructor interface is the same of the ParallelFor (but the template type)
- parallel_reduce method interface
  - ***parallel_reduce(var, identity-val, first, last, step, chunk, mapF, reduceF, nworkers)***
  - ***mapF and reduceF are C++ lambda-functions***

ParallelForReduce *example*

- Dot product (or scalar product or inner product), takes to vectors (A,B) of the same length, it produces in output a single element computed as the sum of the products of the corresponding elements of the two vectors. Example:

```
long s=0;
for(long i=0; i<N; ++i)  s += A[i] * B[i];
```

- Let's comment the FastFlow parallel implementation in the tutorial folder

  <fastflow-dir>/tutorial/fftutorial_source_code/examples/dotprod/dotprod.cpp

ClassWork5: finding prime numbers

- Same problem of ClassWork4.
- Give a parallel implementation of the problem by using the FastFlow ParallelFor pattern.