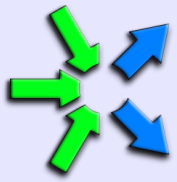
FastFlow introduction (2)



Massimo Torquati

Computer Science Department, University of Pisa – Italy

torquati@di.unipi.it

SPM, 18 November 2014

Data Parallel Patterns

- 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 method 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***

ParallelFor/Reduce example: dot-product

```
float *A = new float[N];
float *B = new float[N];

// initialize A and B
for(long i=0;i<N;++i) A[i] = float(i);
for(long i=1;i<N;++i) B[i] = A[i-1] * float(i);

float dp = 1.0;
for(long i=0; i<N; ++i)
     dp += A[i] * B[i];

std::cout << “dp= “ << dp << “\n”;
```

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

float *A = new float[N];
float *B = new float[N];

ParallelForReduce<float> pfr;

// initialize A and B
pfr.parallel_for(0,N, [&A](const long i) {A[i] = float(i);});
pfr.parallel_for(0,N, [&B, A](const long i) {
     B[i] = A[i-1] * float(i);
   });

float dp = 1.0;
pfr.parallel_reduce(dp, 0.0,
                            0,N,
                            [A,B](const long i, float &dp) {
                                    dp += A[i] * B[i];
                            }, [ ](float &dp, const float e) {
                                    dp += e;
                            });
std::cout << “dp= “ << dp << “\n”;
```

ParallelFor/Reduce simple tests

- Take a look at the simple test **parfor_basic.cpp** contained in the FastFlow folder **tutorial/basic_tests**.