# From C++11 Attributes to Parallel FastFlow Code

## Automatic generation of efficient parallel code by using REPARA attributes and the FastFlow run-time

*University of Pisa and Torino, Italy - REPARA FP7 EU-STREP project*

## What

The whole computer hardware industry embraced parallel platforms, such as multicore, GPGPUs, and cloud. For these platforms, the extreme optimisation of sequential algorithms is no longer enough to squeeze the real machine power. Parallel Patterns provide a mechanism to express parallelism at a high level of abstraction and to make easier the transformation of existing legacy applications for parallel environments.

In the long term writing parallel programs ought to be as efficient, portable, and correct as it has been to write programs for sequential computers. By definition, the raison d'être for parallel computing is high performance, but speed-up need not to be the only measure. Human productivity, total cost and time to solution are equally, if not more important.

## How

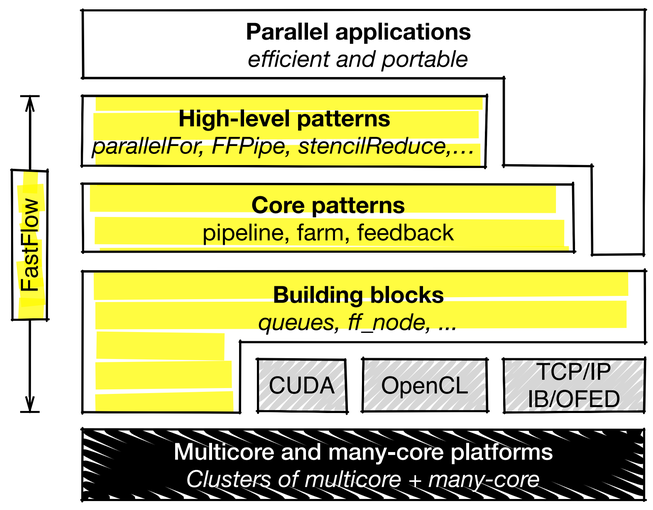
FastFlow is a C++ parallel programming framework advocating high-level, pattern-based parallel programming. It chiefly supports streaming and data parallelism, targeting heterogenous platforms composed of clusters of shared-memory platforms, possibly equipped with computing accelerators. The FastFlow run-time efficiently supports fine grain parallelism via non-blocking multi-threading with lock-less synchronisations zero-copy messaging and asynchronous accelerator offloading.

Almost any x86-64, Arm, PPC system with a C++11 compiler is supported, e.g. Linux, MacOS, Windows. GPGPUs are also supported via either OpenCL or CUDA.

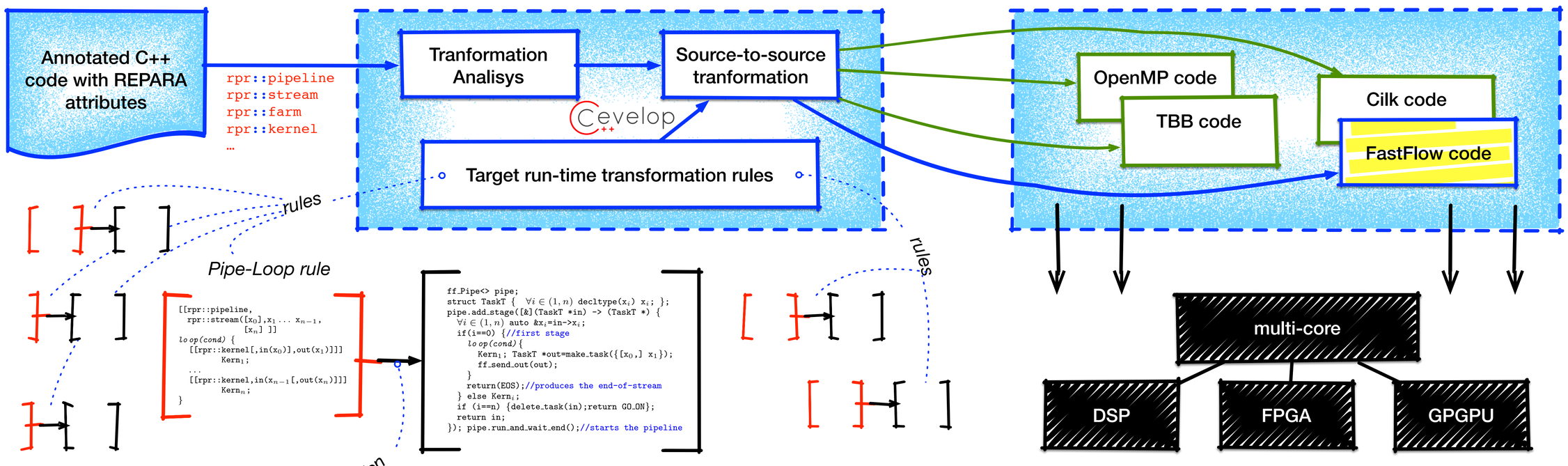
Open source software under LGPLv3

**http://mc-fastflow.sourceforge.net/**

## FastFlow: a layered architecture




## Reengineering and Enabling Performance And poweR of Applications (REPARA) and FastFlow




*REPARA code*

```
int main(int argc, char *argv[]) {
  InitializeMagick(*argv);
  auto s = &argv[1];
  auto nWorkers =32;
  fileNameIterator fit(s);
  std::string filepath;
  Image img;
  [[rpr::pipeline,
    rpr::stream(filepath ,img)]]
  while( fit.getNext(filepath) ) {
    [[rpr::kernel,
      rpr::in(filepath ,img),
      rpr::out(img)]]
    ReadImage(filepath , img);
    [[rpr::kernel,
      rpr::farm(nWorkers),
      rpr::in(img),
      rpr::out(img)]]{
        ApplyFilter1(img);
        ApplyFilter2(img);
    }
    [[rpr::kernel,
      rpr::in(filepath, img)]]
    WriteImage(filepath, img);
  }
  return 0;
}
```

*FastFlow code*

```
struct stream_t {
  stream_t(const std::string &filepath,
           Image &img) :
             filepath(filepath), img(img) {}
  std::string filepath;
  Image img;
};

struct FirstStage: ff_node_t <stream_t > {
  FirstStage(fileNameIterator &fit):fit(fit) {}
  stream_t *svc(stream_t *) {
    std::string filepath;
    while( fit.getNext(filepath) ) {
      ReadImage(filepath ,img);
      stream_t *t = new stream_t(filepath ,img);
      ff_send_out_to(t);
    }
    return EOS; // End-Of-Stream
  }
  fileNameIterator fit;
  Image img;
};

struct Combine: ff_node_t <stream_t > {
  stream_t *svc(stream_t *task) {
    Image &img = task ->img;
    ApplyFilter1(img);
    ApplyFilter2(img);
    return task;
  }
};
```

```
struct LastStage: ff_node_t <stream_t > {
  stream_t *svc(stream_t *task) {
    std::string &filepath = task->filepath;
    Image &img = task ->img;
    WriteImage(filepath ,img);
    delete task; return GO_ON;
  }
};

int main(int argc, char *argv[]) {
  InitializeMagick(*argv);
  auto s= &argv[1];
  auto nWorkers =32;
  fileNameIterator fit(s);
  std::string filepath;
  Image img;
  // builds a vector of farm's workers
  std::vector<std::unique_ptr<ff_node> > W;
  for(size_t i=0;i<nWorkers;++i)
      W.push_back(make_unique <Combine >());
  // farm instance
  ff_Farm <stream_t ,stream_t > Farm(std::move(W));
  // pipeline instance
  ff_Pipe <> Pipe(make_unique <FirstStage >(fit),
                  Farm, make_unique<LastStage>());
  // on-demand task scheduling
  Farm.set_scheduling_ondemand();
  if (Pipe.run_and_wait_end()<0) error("pipe");
  return 0;
}
```

REPARA

SEVENTH FRAMEWORK PROGRAMME

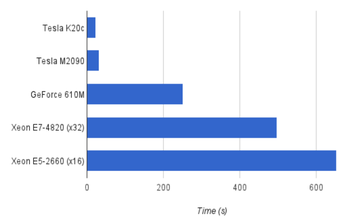
The REPARA FP7 EU-STREP project aims to help the transformation and deployment of new and legacy applications in heterogeneous platforms while maintaining a balance between performance, energy efficiency and code maintainability.

- Exploits C++11 attributes to introduce parallelism in sequential code
- Supports heterogeneous platforms (multi-core, GPGPUs, FPGAs, DSPs)
- Methodology: Sequential code ➡ Annotations ➡ Refactoring ➡ Parallel run-time for multiple CPUs and accelerators
- Objectives: boost performance, reducing power and energy, enhancing time-to-solution
- Partners: University Carlos III Madrid, HSR Rapperswil, Tech. Univ. Darmstadt, Univ. Pisa, Univ. Torino, Univ. Szeged, EVOPRO Innovation, IXION

## Signal filtering

A high-performance filter for real-time video restoration though variational methods.

- Two-stage pipelined filtering: detection and correction
- Correction phase using variational method
- Successfully restoration of up to extreme levels of impulsive and gaussian noises
- Seamless portability to nVidia CUDA K-xx GPGPUs

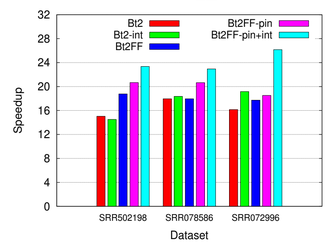




## DNA alignment (Bowtie2 & BWA)

The Bowtie2 and BWA, which are among the fastest and most used alignment tools for genome analysis.

Bowtie2, BWA and several other parallel alignment tools exhibit the same concurrency exploitation pattern (pool of threads working on independent tasks).

- They can be all ported to FastFlow by exploiting the task-farm pattern
- Only few lines of code changed
- Higher performance with decreased development effort


## and much more …

- High-performance deep packet inspection
- High-frequency streaming applications
- High-Performance Haplotype assembly
- Parallel C4.5 classification
- Parallel bzip
- Parallel Smith-Waterman
- Large graph building and analysis
- Video stereo matching
- Parallel Gillespie simulation
- Parallel Numerical Algorithm (Cholesky, LU)
- Parallel C++ memory allocator

**Credits:** M. Danelutto, M. Torquati, M. Aldinucci
Univ. of Pisa and Torino, Italy