# Tutorial of Cilk Plus

## Installing Cilk Plus

Cilk Plus is already available in GCC 5.1 or greater. However, in this tutorial we will show how to install the Cilk branch of GCC 4.8. The reason is that the debugger tool cilkscreen works only with GCC 4.8 and the Intel compilers. So, if you want to use cilkscreen, you have to compile your code with the Cilk branch of GCC 4.8.

For the rest of the tutorial, we will assume that the operative system is Ubuntu.

1. First of all, you need to install the following prerequisites:

```
$  sudo aptitude install flex
$  sudo aptitude install bison
$  sudo aptitude install make
$  sudo aptitude install gcc
$  sudo aptitude install autogen
$  sudo aptitude install cloog-isl
$  sudo aptitude install libcloog-isl-dev
$  sudo aptitude install libc6-dev-i386
```

2. Then, download the Cilk branch GCC 4.8 using HTTP. In the following link, press the *snapshot link*: https://gcc.gnu.org/git/?p=gcc.git;a=shortlog;h=refs/heads/cilkplus-4_8-branch
3. Let's assume that you want to install Cilk in the $TARGET folder. The next steps are:
    1. Uncompressed the Cilk branch: `tar -xf gcc-3cfca5e.tar.gz`
    2. `mv gcc-3cfca5e $TARGET/cilkplus-gcc`
    3. `cd $TARGET/cilkplus-gcc`
    4. Download some extra prerequisites: `./contrib/download_prerequisites`
    5. `mkdir ../b-gcc`
    6. `cd ../b-gcc`
    7. `../cilkplus-gcc/configure --prefix=$TARGET/cilkplus-install -enable-languages="c,c++"`
    8. `make`
    9. `make install`
    10. You can remove the folders cilkplus-gcc and b-gcc

4. We are almost ready. Right now, the Cilk branch is install in $TARGET. To make it available for the entire system, create a file called cilk.sh with the following lines:

```
export PATH=$TARGET/cilkplus-install/bin:$TARGET/cilkplus-install/cilktools/bin:$PATH
export CPATH=$TARGET/cilkplus-install/include:$CPATH
export LIBRARY_PATH=$TARGET/cilkplus-install/lib:$TARGET/cilkplus-install/lib64:$LIBRARY_PATH
export LD_LIBRARY_PATH=$TARGET/cilkplus-install/lib:$TARGET/cilkplus-install/lib64:$LD_LIBRARY_PATH
```

5. And the final step. Download the following file, uncompressed it and put their content in $TARGET/cilkplus-install/cilktools. The file correspond to cilkscreen and other cilk tools. https://www.cilkplus.org/sites/default/files/cilk_tools/cilktools-linux-004501.tgz

6. To test if everything is right, type the following command in your terminal

   ```
   $ source cilk.sh
   ```

   With this command you make available the Cilk branch of GCC 4.8 for the current terminal session. Now, type

   ```
   $ gcc --version
   ```

   The output should be this:

   ```
   gcc (GCC) 4.8.1 20130520 (prerelease)
   Copyright (C) 2013 Free Software Foundation, Inc.
   This is free software; see the source for copying conditions.  There is
   NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
   PURPOSE.
   ```

# Compiling

To compile with GCC Cilk, you have to include the following libraries:

```
#include <cilk/cilk.h>
#include <cilk/cilk_api.h>
#include <cilk/common.h>
```

and include the flags `-fcilkplus` and `-lcilkrts` is your compilation command. For example, if you want to compile the file example.c, the compilation command should be:

```
$ gcc -o example example.c -fcilkplus -lcilkrts
```

# Running your code

To run you code with a fixed number of threads, you have to use the environment variable `CILK_NWORKERS`. For example, if you want to run your parallel code with 4 threads, you have to run the following command

```
$ CILK_NWORKERS=4 ./example
```

Notice that running `./example`, without `CILK_NWORKERS`, will use all the available cores of the machine. As an option, you can set the number of threads into the code, by using the function `__cilkrts_set_param`. For the previous example, we have `__cilkrts_set_param("nworkers", 4)`.

# First example: Fibonacci numbers

The first example is the usage of **cilk_spawn** and **cilk_sync** keywords. In this example, we parallelize the recursive implementation of the Fibonacci function. The implementation is available in the file **fibonacci.c**

# Second example: Image processing

In this second example, we parallelize an algorithm of image processing. In particular, we process the evolution of an image where each pixel depends on its neighbors. We will show that such kind of algorithms can be processed in parallel using the **cilk_for** keyword.

As we discussed in our classes, cilk_for is implemented by dividing the range of iterations until reaching a grain size. By default, the grain size used is $\min(2048, \mathrm{N}/(8*\mathrm{p}))$, where $\mathrm{N}$ is the number of iterations and $\mathrm{p}$ is the number of threads or workers. It is possible to set a different value for the grain size using a pragma instruction in yout Cilk code:

```
#pragma cilk grainsize = expression
```

For example:

`#pragma cilk grainsize = 1` or

`#pragma cilk grainsize = N/(4*__cilkrts_get_nworkers())`, where `__cilkrts_get_nworkers()` corresponds to the number of available threads.

For more details of the implementation of cilk_for, you can review the following link: https://software.intel.com/en-us/node/522593

# Profiling: Perf

Perf is an profiling tool for Linux systems. During this course, perf will help us to measure the number of instructions and the number of cache misses of our programs, in running time. To install perf in your system, you have to run the following command:

Where the option -e allow us to collect information of our system, called events. For a complete list of events, type

```
$ sudo aptitude install linux-tools-generic
```

For us, the most common usage of perf will be

```
$ CILK_NWORKERS=4 perf stat -e instructions,LLC-load-misses,LLC-store-misses ./example
```

Where the option -e allow us to collect information of our system, called events. For a complete list of events, type

```
$ perf list
```

**Note:** Sometimes you need to change the value of the file `/proc/sys/kernel/perf_event_paranoid`. Change it to -1 for a full access