**Section 1: Dataset preparation**

We choose three cases from Phoenix and Splash2(Stanford Parallel Applications for Shared-Memory ).

|  |  |  |
| --- | --- | --- |
| histogram | Phoenix | False sharing |
| linear-regression | Phoenix | False sharing |
| lu\_ncb | Splahs2 | True sharing |

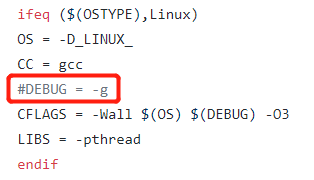
Phoenix Repo: <https://github.com/kozyraki/phoenix/tree/master/phoenix-2.0>

Splash2 Repo: <https://github.com/staceyson/splash2>

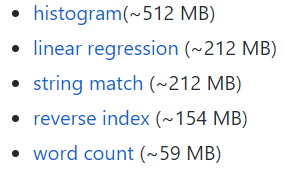
Clone these two repos and directly use make command to generate target binary program.

**Attention:**

1. Remember to add -g in Defines.mk or Makefile:



1. This file <https://github.com/kozyraki/phoenix/blob/master/README.md> provides some input data for testing program:



1. Generated binary programs have different versions, we should test -pthread version, for instance, histogram-pthread.



**Section 2: Deploy Feather environment**

Actually, Feather has been integrated into WitchTools, which is based on HpcToolKit. WitchTools has some other academic works, such as LoadSpy, RedSpy.

To install Feather, we need to install hpctoolkit-externals and libmonitor first.

1. Hpctoolkit-externals

Hpctoolkit-externals includes many components that Feather needs. Its repo is:

<https://github.com/WitchTools/hpctoolkit-externals>

Install procedure:

1. cd to hpctoolkit-externals folder;
2. ./configure
3. make all
4. make distclean

Attention: We don’t need to specify –prefix when run configure, because externals only compile some necessary libraries without “make install”.

With debugging symbol

1. export CXXFLAGS="-g"
2. cd to hpctoolkit-externals folder;
3. ./configure
4. make all
5. make distclean
6. unset CXXFLAGS
7. libmonitor

1. This tool’s repo is: <https://github.com/WitchTools/libmonitor>, clone it and cd to its directory.

2. ./configure --prefix=/path/to/install

3. make

4. make install

With debugging symbol

1. export CPPFLAGS="-g"
2. export CFLAGS="-g"
3. ./configure --prefix=/path/to/install
4. make
5. make install

#Currently ~/hpc\_tools/debug/libmonitor

1. hpctoolkit
2. hpctoolkit’s repo is: <https://github.com/WitchTools/hpctoolkit>, clone it and cd to target directory.
3. Run configure by the following command:

./configure --prefix=/path/to/install/hpctoolkit --with-externals=/path/to/installed/hpctoolkit-externals --with-papi=/path/to/installed/PAPI --with-libmonitor=/path/to/installed/libmonitor

(Attention: Here, we need to install PAPI(<https://github.com/WitchTools/PAPI>). The install procedure is easy and please refer to this file:

<https://github.com/WitchTools/PAPI/blob/master/INSTALL.txt>)

1. make
2. make install

After installing hpctoolkit, we need to run it with programs to be analyzed. First, change directory to installed hpctoolkit. Then, perform the following commands:

(Let us take linear\_regression-pthread as an example.)

1. hpcrun -o <hpcrun\_result\_directory> -e WP\_FALSE\_SHARING -e MEM\_UOPS\_RETIRED:ALL\_STORES@<sampling rate> ./linear\_regression-pthread input\_file

PS: input\_file can be acquired from Section 1: Attention(2).

1. hpcstruct linear\_regression-pthread

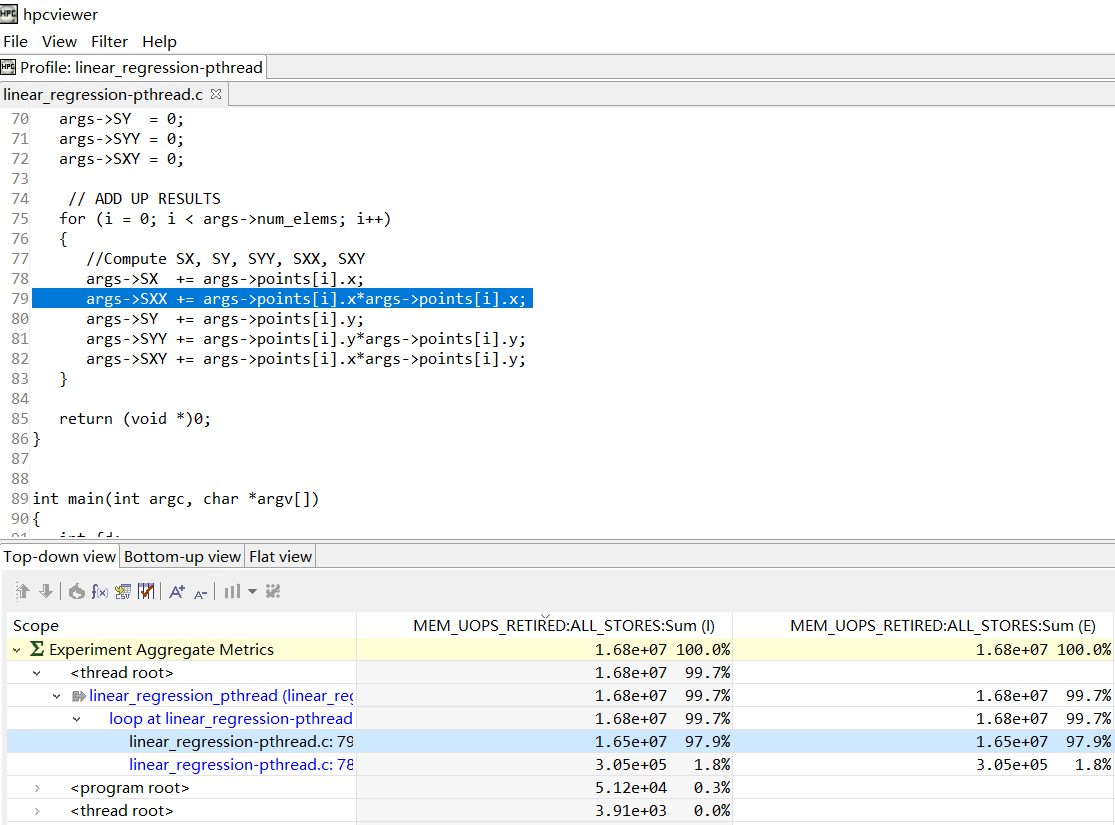
After step 2, current folder will generate a file called linear\_regression-pthread.hpcstruct.

1. hpcprof -S ./linear\_regression-pthread.hpcstruct -o <hpcprof\_result\_directory> <hpcrun\_result\_directory>

For debugging symbol, enable --enable-develop in configure

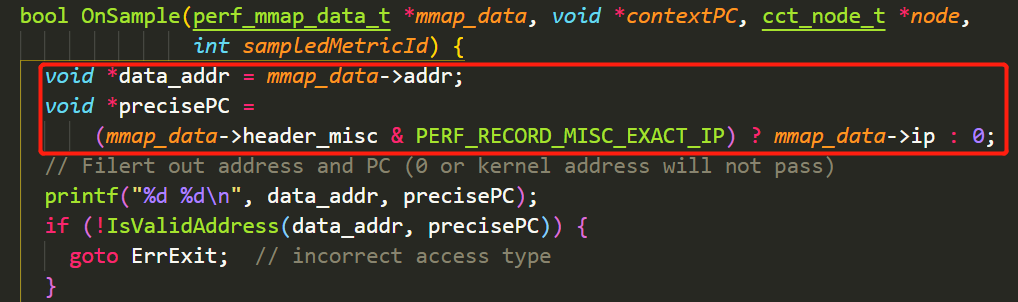
1. hpcviewer

This tool is used to check the results of hpctoolkit. Use hpcviewer to open hpcprof\_result\_directory, then we can check the analysis result:



A Problem about Linux\_Perf:

In watchpoint\_clients.c, which belongs to hpcrun, I found that addr and IP from sample data provided by Linux perf are always equal to 0. The code is here:



Addr and IP are parsed from a mmap space. So, in this condition, we can’t pass the check in function IsValidAddress.

The call chain (They are all in hpcrun section) is: **watchpoint\_clients.c:OnSample -> linux\_perf.c:record\_sample -> watchpoint\_support.c: perf\_event\_handler**.

Code for parsing perf data in perf\_event\_handler is listed in the following figure:

