CPTNHOOK

A tool for scientific application run-time optimization

E. M. Asimakopoulou [1] D. Piparo [2] V. Innocente [2]

August 2, 2015

[1]Department of Physics, Royal Institute of Technology, Stockholm, Sweden

[2]PH-SFT Division, CERN

What is it

A tool, implemented to instrument a scientific application
- applications for *data analysis, reconstruction, simulation*, etc -
with the goal of **optimizing** the application **run-time**.

Motivation

Scientific applications have the pressing need of becoming less resource-hungry. This includes their run-time too.

A current issue is the extensive use of very time-consuming mathematical functions (such as transcendental functions), a result of which is their accounting for a big portion of the programs' run-time

Solving the problem

One way to address this problem is to use **ad hoc approximations** of mathematical functions (e.g. polynomial approximations, Pade's approximation).

To achieve this goal a clear overview of the usage of mathematical functions in the program is required.

... ENTER CPTNHOOK

The tool hooks into functions' arguments whenever they are used. Thus providing an overview of the use of each function (argument values, location of use).

This can later be used as input for improving the run-time of the program.

Example

Assume a mathematical function, sin(x).

During the process of a scientific application a variety of mathematical functions are used, including sin(x).

Assume further that for the purposes of the analysis, sin(x) is used in a specific range of values, ex. $0 - \frac{\pi}{2}$

Example (cont.)

The usual approach for a program is to use reduction steps for the calculation of the function $\rightarrow$ quite costly

One way to improve this is to differentiate the used **argument ranges**, in other words, re-optimizing the “chopping” of ranges for the calculation of the functions.

Example (cont.)

i.e.:
use a different mechanism for the calculation depending on the **frequency** of use of an **argument range** (from a specific function call location):

highly used values → fast mechanism, (fast polynomials)
less frequent values → slow mechanism

Note: Both ranges will have the same precision in their calculation

Example (cont.)

For such a solution plan to exist a **monitoring** of the use of the mathematical functions through out the application is required.

In this regard the present tool has been developed.

CHALLENGES

The main challenges for the development of such tool are twofold:

1. **Instrumenting the program**
   Hooking into the program and getting the required **arguments** without modifying the source
2. **Identifying the location of each function call**
   Just knowing the argument of the function does not provide the needed information.
   It's important to know the **range of values** of a specific function from a certain location in the code, i.e. knowing the **stack trace** of each called mathematical function.

Instrumentation

→ PIN [1]

PIN: A Dynamic Binary Instrumentation Tool

Instrumentation is performed at run time on the compiled binary files

[1] Pin: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

Identification of function call

⋆ Pin: missing the ability to associate call paths with instructions as they execute.

→ Use of **CCTLib** [2]

CCTLib

- call path collection library
- can be used from any Pin tool to obtain full calling context at any and every machine instruction that executes
- associates any instruction with source code along the call path and also points to the data object accessed by the instruction if it is a memory access.

[1]CCTLib: https://github.com/chabbimilind/cctlib

Overcoming the Challenges

- Instrumentation $\longrightarrow$ PIN $\Rightarrow$ get argument values
- Function call location $\longrightarrow$ CCTLib $\Rightarrow$ get stack traces

The outputs of the tool are:

- the arguments values of the called functions
- the location of each argument call for each function (stack trace)

Output

The results are given in ROOT TFile format.

↪ create **TTrees** for each function, with ***branches*** containing:

- the values and
- the hashing of the respective function's stack trace.

↪ make a **vector** containing the ***mapping*** of stack traces and hashes of the mathematical functions.

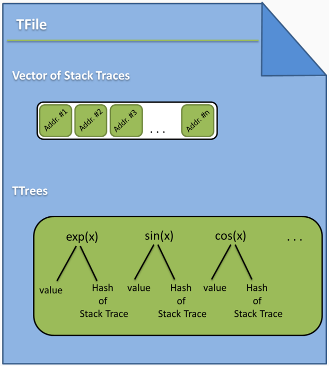

Figure: Root TFile format.

Using cptnHook

The tool is easy to use.
cptnHook can be downloaded from github:

https://github.com/emyrto/cptnHook

and it can be used in 3 simple steps:

1. Ensure ROOT is installed,
2. run "getStarted.sh" → installs PIN, compiles and creates the "cptnHook.so" library

```
sh getStarted.sh
```

3. and use the tool to instrument an executable:

```
cptnHook.py -o myResults ./myExe
```

Future Work

The tool is available and ready to use, however it's still in beta stage.

Further development will be focused on aspects like:

- covering both double and single precision,
- presentation of results in a much more user-friendly way:
  ↪ present separate panels for each function, containing:
  - list of stacks and
  - the respective histogram of argument values.
- (provide the option of modifying the result of the function -reduce it's precision- in order to compare run-time results)

Future Work

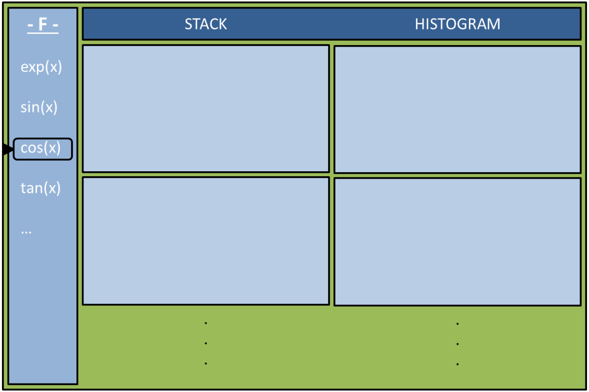

Figure: Presentation of results in later version of the tool.

Summary

cptnHook

- tool that instruments scientific applications
- ***Goal***: reduce run-time

How:

- monitor argument values and stack traces of the mathematical functions in a scientific application (data analysis, reconstruction, simulation, etc)
- provide overview of their use
- assist in implementing a more optimal mechanism in their calculation
- reduce the time consumption of those calculations.