MCProf: Memory and Communication Profiler

Imran Ashraf
Computer Science and Engineering
Department of Software and Computer Technology
Delft University of Technology, Delft, The Netherlands
I.Ashraf@tudelft.nl

November 1, 2014

1 Introduction

MCProf is a memory and communication profiler. It traces memory reads/writes and reports memory accesses by various functions in the application as well as the data-communication between functions. The information is obtained by performing dynamic binary instrumentation by utilizing Intel Pin [1] framework. This manual explains the process of setting up *MCProf* and using it.

2 Availability

MCProf can be downloaded from ...

3 Required Packages

In order to setup and use *MCProf* the following two packages are required:

- Intel Pin DBI framework [1] Revision 62732 or higher
- g++ compiler with support for C++11X
- graphviz Dot utility for converting the generated communication graphs from DOT to pdf formats

4 Installation

MCProf uses Makefile to compile the sources. In order to compile *MCProf* from sources on 32-bit / 64-bit Linux, the following steps can be performed.

- Download Pin and copy and extract it to the directory where you want to keep Pin.
- Define a variable `PIN_ROOT` by running the following commands:

```
export PIN_ROOT=/<absolute path to pin>
```

- You can also add this line, for instance, to your **.bashrc** in case you are using **bash** to export the variable automatically on opening a terminal.
- Download *MCProf* and copy and extract it to the directory where you want to compile it.
- Go the *MCProf* directory and run the following command to compile:

```
make
```

If every thing goes fine, you will see a directory `obj-intel64` (or `obj-ia32` depending upon your architecture). This directory will contain the executables and object files generated as a result of the compilation. The important files are:

- **mcprof.so** which is the tool. This will be used to profile the applications as explained in Section 5.
- executable files of the test applications available in **tests** directory. These executables can be used as test inputs.

5 Usage

In order to explain the usage of *MCProf* we will use the example application listed in Figure 1. The source code is available in `tests` directory of source package.

```
1  int *srcArr1, *srcArr2, *sumArr, *diffArr;
2  int coeff = 2;
3  int nElem;
4
5  void initVecs(){
6      for(int i = 0; i < nElem; i++){
7          srcArr1[i]=i*5 + 7;
8          srcArr2[i]=2*i - 3;
9      }
10 }
11 void sumVecs(){
12     for(int i = 0; i < nElem; i++)
13         sumArr[i] = srcArr1[i] + coeff * srcArr2[i];
14 }
15 void diffVecs(){
16     for(int i = 0; i < nElem; i++)
17         diffArr[i] = coeff * (srcArr1[i] - srcArr2[i]);
18 }
19
20 int main(){
21     nElem = 100;
22
23     srcArr1 = malloc(nElem*sizeof(TYPE));
24     srcArr2 = malloc(nElem*sizeof(TYPE));
25     sumArr  = malloc(nElem*sizeof(TYPE));
26     diffArr = malloc(nElem*sizeof(TYPE));
27
28     initVecs();
29     sumVecs();
30     diffVecs();
31
32     free(srcArr1);
33     free(srcArr2);
34     free(sumArr);
35     free(diffArr);
36
37     return 0;
38 }
```

Figure 1: Example of an application processing some arrays.

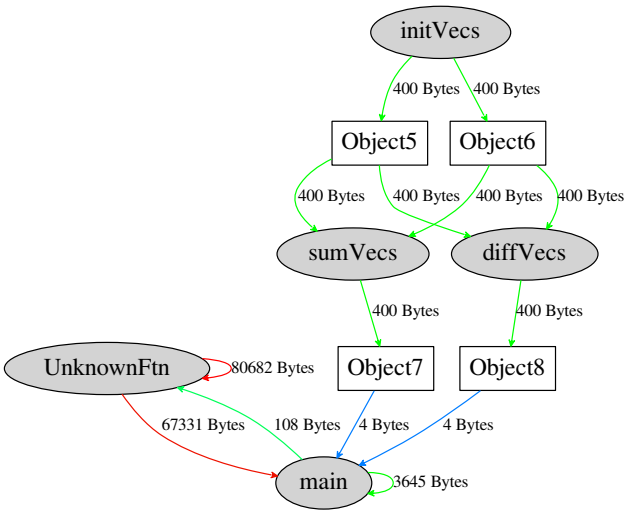

Figure 2: Figure 1 caption.

5.1 Using Given Tests

This example will be compiled during the default compilation of the *MCProf* discussed in Section 4. In order to profile this application by *MCProf* you can give the following command:

```
make vectOps.test
```

Simmilarly, other tests can also be executed by replacing the <vectOps> with the <name of the test application>.test as given in **tests** directory. This will generate the output information depending upon which engines is used. This will be explained in Section 6.

5.2 Using Your Own Example

This application can be compiled

6 Output

Figure ?? shows the output communication graph.

7 Frequently Encountered Problems

This section will cover some of the frequently encountered problems will setting-up/using *MCProf*.

7.1 Pin Injection Mode Error

On some systems if Pin (parent) injection mode is not enabled by default then you see an error as shown below.

```
E:Attach to pid 13972 failed.
E:  The Operating System configuration prevents Pin
E:  from using the default (parent) injection mode.
E:  To resolve, either execute the following (as root):
E:  $ echo 0 > /proc/sys/kernel/yama/ptrace_scope
E:  Or use the "-injection child" option.
E:  For more information, regarding child injection,
E:  see Injection section in the Pin User Manual.
```

Solution is also suggested in this message, which is to become root and enable this injection. This can be achieved by running the following two commands:

```
sudo -i
echo 0 > /proc/sys/kernel/yama/ptrace_scope
exit
```

8 Contact

In case you are interested in contributing to *MCProf*, or you have suggestions for improvements, or you want to report a bug, contact:

- Imran Ashraf < I.Ashraf@TUDelft.nl >

References

[1] C.K. Luk and et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In *PLDI '05*, pages 190–200, New York, NY, USA, 2005. ACM.