X-Ray:
Automatic Measurement of Hardware Parameters

`http://iss.cs.cornell.edu/Software/X-Ray.aspx`

Kamen Yotov
`kamen@yotov.org`

February 6, 2005

1 Introduction

This document explains how to use the X-Ray framework [1]. It does not provide a detailed explanation on how X-Ray works internally, which is covered in the X-Ray papers [2, 3]. It is my hope that over time, the current document will contain all the relevant information about X-Ray.

2 Installation

X-Ray is available through CVS repository root `:pserver:cvs@iss.cs.cornell.edu:/software`, module `x-ray`, branch `b2`. Work has started on X-Ray 2.0 and this document will describe only this newer version. Note that instruction for use are very different for prior versions. You can get the latest copy using the following command line commands:

1. `cvs -d :pserver:cvs@iss.cs.cornell.edu:/software login` – this logs you in to the Intelligent Software Systems CVS server. When asked for a password enter "cvs".
2. `cvs -d :pserver:cvs@iss.cs.cornell.edu:/software co -r b2 x-ray` – note the `-r b2`, which is essential. If you skip it the command will complete successfully but you will get a copy of X-Ray 1.0.
3. Pick a name for your machine/platform (e.g. foo) and create a file called `foo.platform` in the root X-Ray folder using `generic.platform` as an example. Typically this involves specifying the native C compiler on the platform (with a reasonably high optimization level), and utilities to copy and delete files (alternatives to the UNIX `cp` and `rm`).

3 Execution

X-Ray is a framework in which one implements micro-benchmarks. A micro-benchmark determines a hardware parameter by running experiments on the target machine. We call these experiments nano-benchmarks. A detailed description of micro- and nano-benchmarks and their relation to X-Ray is given in [2].

3.1 Parameters

Benchmarks can accept parameters. For example one parameter for the micro-benchmark for determining the number of registers is the type of registers to be determined (i.e. integer, double, SSE2, etc.) X-Ray distinguishes between two types of parameters – *code parameters* and *data parameters*.

- The code of the benchmark depends on the code parameters, and therefore a change of the value of a code parameter requires code generation and recompilation. For example, consider the nano-benchmark for measuring the latency of an instruction. The operation O to be measured is a code parameter for this nano-benchmark because different code needs to be generated for different operations O.
- Data parameters are parameters which are passed to the already compiled benchmark executable without the need for recompilation. For example, consider the description of address sequences used to determine the parameters of the data cache. They are only used to initialize the memory array before measurement and therefore do not require recompilation of the code.

X-Ray distinguishes between code and data parameters for reuse purposes. It compiles one executable for each set of code parameters for every benchmark. It then is able to reuse this executable for measuring with different data parameters. Finally, all results of running benchmarks are cached in the file `out/storage.txt` as a lookup table. The key in this table is the name of the benchmark coupled with all supplied parameters (code and data).

3.2 Executing a benchmark

To execute a benchmark, issue the following command from the x-ray root directory:

```
gmake run TYPE=<type> NAME=<name> <parameter1>=<value1> ...
```

Note that you can supply as many parameters as needed. The first two have special meaning. `<type>` can only be `MBM` (micro-benchmark) or `NBM` (nano-benchmark). `<name>` is the name of the corresponding benchmark. If you

supply at least these two parameters X-Ray will remind you what are the other parameters needed to perform this benchmark.

```
% gmake run TYPE=MBM NAME=latency
** ERROR: Missing parameters:  TYPE=[MBM] NAME=[latency] O=[] T=[]
```

This example shows that to run the "latency" micro-benchmark one needs to specify an operation O and type T. Here is one correct invocation:

```
% gmake run TYPE=MBM NAME=latency O=ADD T=F32
```

Normally this will print nothing, but compute and store the result in `out/storage.txt`. There are two special parameters you can use to modify this default behavior. If you specify `VERBOSE=1` measured results are also printed on the screen:

```
% gmake run TYPE=MBM NAME=latency O=ADD T=F32 VERBOSE=1
** nbm_latency_O_ADD__T_F32 -> 2.678
** nbm_latency_O_ADD__T_I32 -> 0.672
** mbm_latency_O_ADD__T_F32 -> 3.984
```

Notice that the first two results are from nano-benchmarks executed as part of the latency micro-benchmark. The first one has measured the latency of floating point add as 2.678 nanoseconds and the second one that of integer add (which we assume is the clock cycle of the CPU) as 0.672 nano-seconds. The micro-benchmark then divides the two to obtain the latency of floating point add in clock cycles ($3.984 \approx 4$ in this case).

If you want even more output, you can specify `OUTPUT=1`, in which case you will also see the invocations of the compiler with the appropriate parameters.

The parameters of all benchmarks (micro and nano) are described in Section 4.

4 Included Benchmarks Description

This section provides a very short reference of the meaning of the parameters required for the different micro- and nano-benchmarks. I will not provide a detailed description of the benchmarks for now. If you need it, please consult the papers [2, 3].

4.1 Micro-benchmarks

4.1.1 mhz

Measures CPU frequency.

Parameters: none.

4.1.2 latency

Measures instruction latency.
Parameters:

- O: instruction (operation), e.g. ADD, MULTIPLY, etc.
- T: type of operands, e.g. I32, F32, F64, etc.

4.1.3 throughput

Measures instruction throughput.
Parameters: same as in "latency".

4.1.4 registers

Measures the number of registers available for allocating variables into them.
Parameters:

- T: type of registers to measure, e.g. I32, F32, F64, etc.

4.1.5 fma

Checks for the presence of dedicated hardware for executing fused multiply-add instructions.
Parameters:

- T: type of operands to check for, e.g. I32, F32, F64, etc.

4.1.6 cache_sa

Measures the cache stride and the cache associativity. Please read [3] to get a complete understanding of the meaning of the parameters.
Parameters:

- s: stride for the higher cache level. Use `sizeof(void *)` for measuring parameters of the L1 data cache.
- a: associativity for the higher cache level. Use 1 for measuring parameters for the L1 data cache.
- P: a big prime number to be used for shuffling the addresses in a sequence. For measuring the parameters of the L1 data cache, 9973 is enough.
- V: because this micro-benchmark measures both cache stride and cache associativity, while the X-Ray framework only allows one result per micro-benchmark, this parameter specifies which result to return. If you specify `V=S` stride is returned. If you specify `V=A` associativity is returned. Note that running the micro-benchmark twice to get both values is not going to double the runtime, because all intermediate measurements are cached.

- L: cache level being measured. This is only symbolic to differentiate the cached results. Use 1 for measuring the parameters of the L1 data cache.

4.1.7 cache_c

Measures the cache capacity.

Parameters: same as "cache_sa" except for V which is obviously not needed here.

4.1.8 cache_b

Measures the cache block (line) size.

Parameters: same as "cache_sa" except for V which is obviously not needed here.

4.1.9 cache_l

Measures the cache hit latency in cycles.

Parameters: same as "cache_sa" except for V which is obviously not needed here.

I have yet to write a complete script for measuring all cache levels automatically. Currently to do all cache levels you need to execute the aforementioned micro-benchmarks manually several times.

Also, I have not yet folded the OS support for super-pages in X-Ray 2.0, so you will have to wait for L2 and below anyway.

4.2 Nano-benchmarks

This will have to wait.

5 Extending X-Ray

In this section I am planning to explain how to extend X-Ray, including how to specify new operations, how to create new micro- and nano-benchmarks, etc.

References

[1] X-Ray: Automatic measurement of hardware parameters. `http://iss.cs.cornell.edu/Software/X-Ray.aspx`.

[2] Kamen Yotov, Keshav Pingali, and Paul Stodghill. Automatic measurement of hardware parameters for embedded processors. Technical Report TR2005-1974, January 2005.

[3] Kamen Yotov, Keshav Pingali, and Paul Stodghill. Automatic measurement of memory hierarchy parameters. In *International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS), Banff, Canada*, 2005.