The New Ithaca and Sampaka Clusters, plus the Flexible Instrumentation Toolkit

Vince Weaver

9 February 2005

The Ithaca Cluster

- 16 Dell 1-U Systems
- Each has two Pentium 4 3.0GHz CPUs
- 1GB of RAM for each node
- two 30GB SCSI drives each node
- Connected by 100Mb/s ethernet
- Running OpenMosix on top of SuSE 9.1
- Open to all of CSL

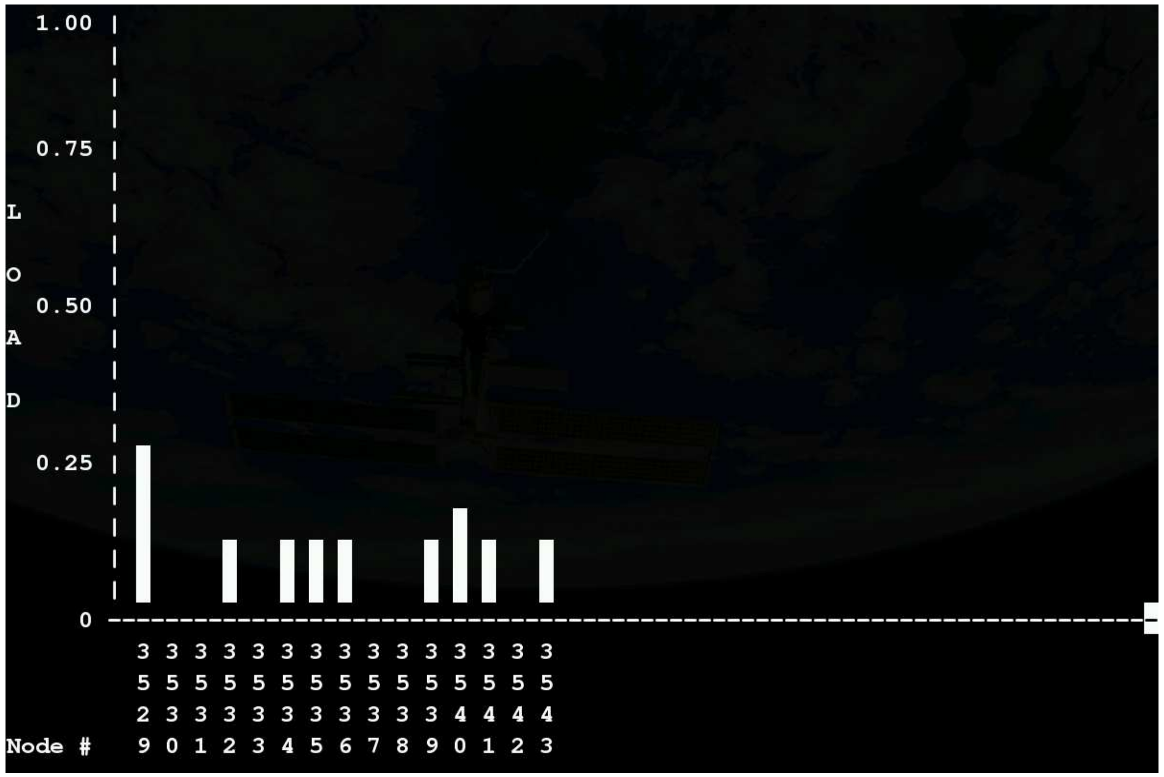
OpenMosix

- http://openmosix.sf.net
- OpenMosix is a clustering system that runs on top of Linux and automatically balances load between nodes
- Process migration algorithm factors in memory pressure, CPU load, and I/O concerns
- Works by running all syscalls on the original node the program is started on. If process is migrated, all syscalls are trapped and passed over the network to the originating node.
- Some programs cannot be migrated. OpenMosix should detect this properly and not migrate in that case.

Using Ithaca

- ssh to ithaca.csl.cornell.edu using your normal CSL account
- Once logged into ithaca, you will need to further ssh into one of the subnodes ithaca01 - ithaca15.
- You can use the “mosmon” utility to see the load across the whole cluster.
- The “mosrun” command lets you specify options on node preference and how you want your process to migrate.
- While in theory you can start as many processes as you want on one node and things will be balanced, it is probably best to limit to 2 per node, especially if you will be doing a lot of I/O.

Sample mosmon Output



The Sampaka Cluster

- 40 1-U Systems
- Each has two Pentium 4 2.8GHz CPUs
- 2GB of RAM for each node
- No disks
- Connected by 1000Mb/s ethernet
- SuSE 9.1 booted over the network
- Eventually limited access

Using Sampaka

- ssh into sampaka.csl.cornell.edu using regular CSL account
- Very soon there will be a batch scheduler. For now just ssh into an individual node (sampaka00-sampaka18).
- As there are no disks, there is no swap. So be careful if your programs use 2+GB of memory. In theory Linux will handle this properly, but due to memory over-commit there are some issues.

FIT - The Flexible Instrumentation Toolkit

- http://www.elis.ugent.be/fit/
- **The design and implementation of FIT: a flexible instrumentation toolkit.** *Bruno De Bus, Dominique Chanet, Bjorn De Sutter, Ludo Van Put, Koen De Bosschere,* Proceedings of the ACM-SIGPLAN-SIGSOFT workshop on Program Analysis for Software Tools and Engineering, pp 29-34, June 2004.
- "FIT is an ATOM-like instrumentation toolkit. It allows you to specify arbitrary instrumentation routines, and produces a custom instrumentor that can statically apply this instrumentation to programs."

FIT - Features

- Compatible with ATOM analysis and instrumentation files
- Works on Linux x86 and Linux ARM. Alpha (Linux, Tru64), IA64, PowerPC and MIPS ports are supposedly underway.
- Open-source, available under the GPL.

FIT - Sample Instrumentation File

```
void InstrumentInit(int argc, char **argv) {
    AddCallProto("BeforeProgram()");
    AddCallProto("AfterProgram()");
    AddCallProto("BeforeLoadStore(long,VALUE)");
}

void Instrument(int argc, char **argv, Obj *obj) {

    Proc *proc; Block *block; Inst *inst;

    AddCallProgram(ProgramBefore, "BeforeProgram");
    AddCallProgram(ProgramAfter, "AfterProgram");

    for(proc=GetFirstObjProc(obj); proc; proc=GetNextProc(proc)) {
       for(block=GetFirstBlock(proc); block; block=GetNextBlock(block)) {
          for(inst=GetFirstInst(block);inst;inst=GetNextInst(inst)) {

             if (IsInstType(inst,InstTypeLoad)) {
                AddCallInst(inst,InstBefore, "BeforeLoadStore",0,EffLoadAddrValue);
             }
          }
```

FIT - Sample Analysis File

```
void BeforeProgram() {
   fd=open("/tmp/trace",O_WRONLY | O_CREAT);
}

void AfterProgram() {
   close(fd);
}

void BeforeLoadStore( long type, long value) {
   buffer[0]=ADDRESS;
   buffer[1]=value;
   write(fd,buffer,2*sizeof(long));
}
```

Instrumentation Parameters

- Can add instrumentation before or after Program, Procedure, Basic Block, or Instruction
- Values you can pass to the analysis routine are:
  - ◇ char or int
  - ◇ REGV: which is the current value of one of the CPU registers (including program counter and stack pointer)
  - ◇ VALUE: Function Name
  - ◇ VALUE: CondValue/BrCondValue - which returns whether the branch instruction's condition code is true or false.
  - ◇ VALUE: EffectiveLoad/Store/Address: Returns the effective address of a load or store instruction. In case of x86 where there can be *both* an instrumentation error is caused.

C that can be used in Analysis File

- Most standard C can be used
- **File I/O**: open, close, fopen, fclose, write, fwrite. These are not buffered so you might want to buffer yourself.
- **String Operations**: strlen, memset, memcpy, strcpy, strncpy
- **printf**: printf, sprintf, fprintf. Some of the more obscure printf features not implemented.
- **malloc**: malloc, calloc, realloc, free. These allocate away from the heap to try to avoid gratuitous address changes. They do use mmap() so can change mmap() addresses. It might be better just to statically allocate regions at compile time.
- FIT does support allocating a static sized array based on data collected at instrumentation time. Somewhat complicated see the README.

Precision Issues with Instrumentation

The addition of instrumentation code causes differences in the run-time execution. There are two major causes:

- **Intrusion on runtime data structures**: If you re-use the program's C-library, and you open a file, this file will be in the opened file list and will make the structure bigger as well as take longer when traversing.
- **Dependence on Data Addresses**: A simple example of this is if a program prints the address of the main() function. This will be different in the instrumented version. (gcc in spec95 apparently used things like the address of *main()* as part of a hash, so can be a real problem).

Drawbacks of ATOM

- ATOM's C-library shares data with the program's C-library. This causes differences in output. FIT addresses this by having its own minimal C-library located elsewhere in memory.

- ATOM changes the addresses used in the original program. FIT goes through great lengths to preserve addresses.

- ATOM claims to be portable. Despite this claim it makes many assumptions that are Alpha and Tru64 specific. For example, it assumes there is a large area between code and data segments where it stores information. On Linux this is not the case, so the address of data changed if code size is made bigger.

FIT Design

- Built on top of Diablo, an optimizing linker. It reads and writes object files, and can disassemble and re-assemble code, and generate control flow graphs.
- The actual "fit" program compiles the instrumentation and analysis code, and creates an instrumentation program.
- This created instrumentation program is then run on the binary, it conjunction with a map file specifying object locations, and in the end the instrumented binary is created.
- To port to a new architecture, a new Diablo backend must be written, and some architectural support code in FIT. As much code as possible is kept generic to try to make porting easier.
- FIT Does not rely on special operating system support or architecture-specific features.

FIT - Precision

FIT attempts to be much more precise than atom.

- **Avoiding shared C-library structures**. FIT has its own custom C-library.
- **Exact memory addresses**. FIT by default does this with ordinary translation. Whenever a load or store address is instrumented, it is translated back to the original address. This is hard to do properly.
- **More exact addresses** FIT is capable of even more exact memory addresses. It does this by keeping all of the original memory addresses and indirect jumps *exactly* the same as the original program. This comes at a price; every load/store/indirect jump has to be translated. Also it requires a kernel patch, because otherwise syscalls with pointers will go to the wrong place.

FIT - Precision Example

- original code:

```
add  r2,r3,r4      ; r2 := r3+r4
load r1,10(r2)     ; load the value of r2+10 into r1
```

- instrumentation code (ordinary translation)

```
add  r2,r3,r4
r1 = translateToOriginal(r2+10)
call Cache(r1)
load r1,10(r2)
```

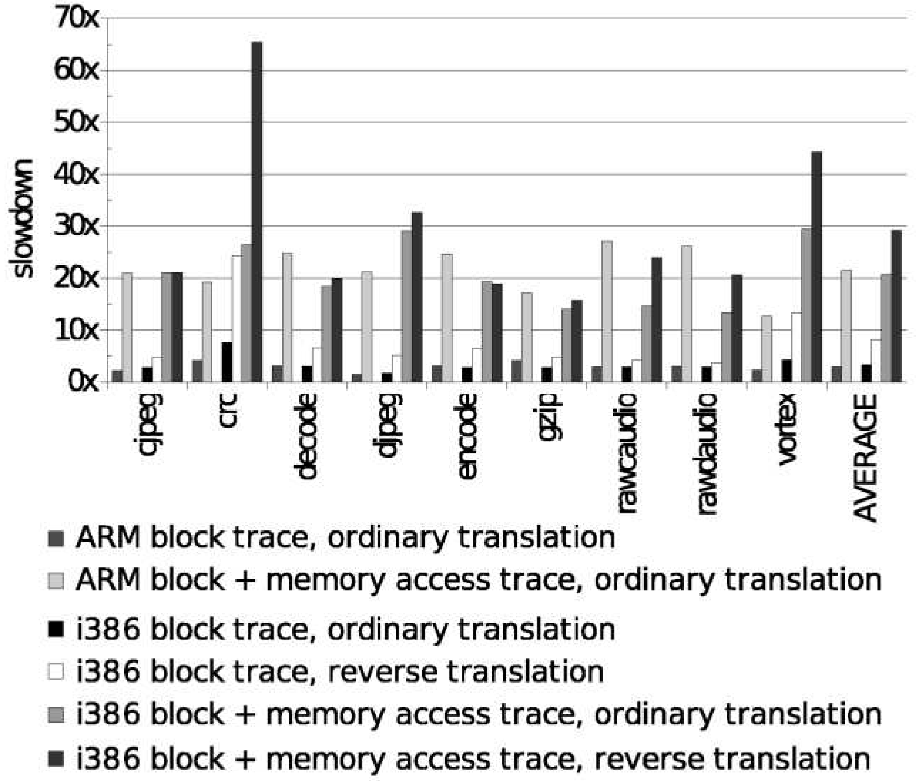
- instrumentation code (reverse translation)

```
add  r2,r3,r4
call Cache(r2+10)
r2 = translateToNewAddress(r2)
load r1,10(r2)
```

FIT - Downsides compared to ATOM

- Instrumentation takes a long time.
- Binaries must me built with a modified toolchain. Code isn't changed; just extra linker layout info is generated. The authors claim they are working on eliminating this step.
- All object files have to be available in the location they were compiled.
- Binaries must be statically linked against libraries.

FIT - Performance



FIT - Actual Usage Report

- Needs a lot of RAM to instrument. It is basically unusable to instrument a program if you have less than 512MB of RAM.
- Instrumenting also takes a while, even with enough RAM. Instrumenting every load and store in equake takes almost 10 minutes on a 2GHz Pentium 4.
- Authors *were* fairly good about responding to patches and bug reports. Currently have gotten no responses from them since before Winter Break.

FIT - Changes I had to make to run 575 Code

- Change “long” to “int”. I actually patched my copy of fit to just treat longs as ints (this works on x86), but the main codebase is waiting to do it right for 64bit architectures.
- Some name functions like “GetProcName()” weren’t implemented. Mainly cosmetic. I sent a few patches in and they were accepted.
- InstrumentFini() is not run at the end of instrumentation. They had reasons why this was hard to do, so not implemented yet. You can work around this by just putting things that would have gone there just at the end of the Instrument() function.

FIT - Tricks to Get Info I Needed

For my current work, in addition to memory traces I needed the size and location of all malloc()s.

- First I had to find and instrument the malloc(), calloc(), realloc() and free() functions. So I had to implement the "get_named_proc()" function, and also figure out the name of the symbols (__libc_malloc).
- Next I had to figure out the parameters to the functions and the return value. This required x86 knowledge. My code passed ESP before and EAX after to analysis routines. Using ESP you can get the parameters off the stack, and after you can find the return value in EAX.
- FIT isn't expecting you to grab parameters off the stack, so then I had to translate back to the original addresses.
- Finally, wanted the call sites of all the malloc calls. So added a way to instrument "call" instructions, and then gathered addresses of all instructions that call malloc().

FIT - Conclusion

- FIT is a valuable new tool.
- Being able to do ATOM-like activities on x86 hardware under Linux is *incredibly* convenient.
- FIT and the patched toolchain are installed on sampaka under */opt/tools/fit*.