PopSPY: A PowerPC Instrumentation Tool for Multiprocessor Simulation

C. Limousin, A. Vartanian, and J-L. Béchennec

Équipe Architectures Parallèles
Université de Paris-Sud XI
Laboratoire de Recherche en Informatique, Bât. 490
91405 Orsay Cedex, France
(limousin,alex,jlb)@lri.fr
tel: 33 1 69 15 64 07
fax: 33 1 69 15 65 86

Abstract. PopSPY is an instrumentation tool for the PowerPC microprocessor family that modifies code at the executable level, providing the ability to instrument a wide range of applications including commercial programs. PopSPY instruments sequential, multithreaded and multiprocess applications including their associated dynamic libraries. PopSPY uses a dynamic trace library that either generates execution traces of applications or communicates with an execution driven simulator for accurate multiprocessor simulation.

Topic 02: Performance Evaluation and Prediction

1 Introduction

Trace driven simulation is a commonly used technique for performance evaluation of architectures. The benchmarks are first run on a real architecture (the *host architecture*) where they are monitored to obtain their execution trace[1], which is then injected in the simulator. Traces are valid for applications whose execution doesn't depend on the architecture they are running on: this is true for most of the sequential applications and for parallel applications that statically allocate shared resources, if the spins at the synchronisation points are not recorded [2]. Execution driven simulation [1] solves this problem enforcing the execution order of the synchronization events on the host architecture to be the same as on the target architecture.

Application monitoring to obtain execution traces is commonly done using a code instrumentation technique, which consists in modifying an application by inserting tracing code sequences around memory instructions and basic blocks. The instrumentation may be done at the assembly source level ([2]), at the object-module level ([7]) or at the executable level ([5], [8], [3]), where one doesn't need the application source or object code.

[1] The execution trace of an application contains the list of executed instructions, addresses of memory accesses, etc...

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 262–265, 1999.

We wanted to study the memory hierarchy (that is to simulate a processor and focus on caches) of mainstream computers architectures, namely PowerPC or x86 (multi or monoprocessors). On those platforms, representative benchmarks are mainly commercial application whose code sources or objects are not available. We thus developped PopSPY, a tool for the PowerPC microprocessor that monitors sequential and shared memory multithreaded applications at the executable level[2]. Since we restrict our research with computing-intensive applications, the fact that the operating system is not monitored is not invalidant. PopSPY uses a shared tracing library that either generates the execution trace of the instrumented applications or is plugged to an execution driven simulator. Moreover, dynamic libraries are also monitored. PopSPY runs under MacOS 7.x or later.

Section 2 presents the functionalities of PopSPY. Performance of PopSPY is exposed in section 3.

2 PopSPY Overview

Working scheme The PopSPY code instrumentor annotates the monitored application with function calls to a shared library: when the annotated application executes, these calls generate the execution trace of the application. Since the library is dynamic, one may rewrite the functions for different purposes without reinstrumenting the application. Only the prototypes of the functions are fixed. The library consists in two main functions, `TraceBlock` and `TraceLdSt`. The first one is called in front of each basic block. It receives the calling block number as a parameter. The second one, which calls are added in front of each memory access, receives three parameters: the calling block number, the number of the load-store instruction in this block and a coded description of this instruction (type, used registers, address compute mode, etc...).

Sequential applications traces A sequential application trace is composed of three files: a file (*static file*) generated at instrumentation time containing the application original basic blocks description (size, memory instructions count, etc...); a file (*dynamic file A*) generated by the library at execution time containing the numbers of executed original basic blocks. These numbers are indexes in the static file; a file (*dynamic file B*) generated by the library containing the description of executed memory accesses (kind, address and data read or written).

With the description of each block a simulator obtains the whole trace from the two dynamic files: the bloc number in the dynamic file A gives the instruction block to be read in the application executable and each time a load-store instruction is encountered, it is read in the dynamic file B.

Shared-memory multithreaded applications traces Contrary to sequential applications a multithreaded one is made of several threads that call the library in

[2] IDTrace [4] for the i486 is compiler-dependant and does not monitor multithreaded applications.

an interleaved way. The library must generate one trace per thread and thus be able to know which thread called it. The multiprocessor support library for MacOS, Daystar MP, gives this information to the trace library. In a multithreaded application trace generated by PopSPY, one will found the same static file as in the sequential case, plus a dynamic file A_i and a dynamic file B_i for each thread i. These A_i and B_i files are generated the same way as in the sequential case. However, an other information must be present in the trace in order to correctly simulate a multithreaded application: the multiprocessor events concerning thread i are interleaved with the block numbers in the A_i file. Supported multiprocessor events are the following: thread creation/merging (fork/join), thread termination, locks management (creation, acquisition, release, destruction), Barrier management.

Multiprocess and dynamic libraries traces PopSPY may add at the user choice an identifier to block numbers passed to the trace library. Several applications with different identifiers may then be executed in parallel: their traces are interleaved in the global trace, each traced instruction being tagged with the idenfifier, allowing a simulator to read correctly each traces. A dynamic library is traced the same way: it is given a unique identifier, and its trace is interleaved with the application one.

Execution driven simulation When a simulator is connected to PopSPY via the trace library, the `TraceBlock` and `TraceLdSt` functions send the informations they receive to the simulator. The trace library intercepts the calls from the application to the multithread support library. Whenever an event whose execution order may modify the trace occurs, the trace library freezes the corresponding threads. The simulator executes these events then feed back the trace library with temporal informations. The trace library uses these informations to release the threads in the simulation order.

3 PopSPY Performance

Examples of applications known to run properly after being instrumented by PopSPY are all programs compiled with Metrowerks Codewarrior, and particularly the benchmark suite SPEC95 [6] and the parallel suite SPLASH-2 [9]. Adobe Photoshop 4.0.1 Tryout and Strata StudioPro Blitz 1.75 which are multithreaded image creation programs are also properly executing after instrumentation.

Measuring the execution of *Adobe Photoshop 4.0.1 Tryout* resizing an image from 72 to 300 dpi, *Strata StudioPro Blitz v1.75* during the Phong rendering of a texture mapped sphere and the five benchmarks *compress*, *go*, *hydro2d*, *mgrid* and *tomcatv* from the SPEC95 suite, executed with the standard data sets on a 90Mhz PowerPC 601 under MacOS 8.0 shows a memory dilation (resp. slowdown) between 3.02 and 6.59 (resp. 2.8 and 16.37), depending of the benchmark and the completeness of instrumentation. As a comparison, Pixie and Goblin have memory dilation of respectively 6 and 4 and slowdowns of 10 and 20.

4 Conclusion

We presented in this paper an instrumentation tool for the PowerPC microprocessors family called PopSPY. This tool instruments applications at the executable level and allowed us to instrument commercial applications like *Adobe Photoshop Tryout* and *Strata StudioPro Blitz*. PopSPY monitors sequential, multithreaded (such as SPLASH-2) and multiprocess applications, including dynamic libraries. Moreover, the dynamic trace library of PopSPY may be plugged to an execution-driven simulator.

We saw that the slowdown and memory dilation of PopSPY are in the order of magnitude of existing executable-level instrumentation tools, making it fully usable. Moreover, a graphical user interface is provided. An extended version of this paper can be found at http://www.lri.fr/∼limousin/biblio.html.

References

[1] R.C. Covington, S. Madala, V. Mehta, and J.B. Sinclair. The rice parallel processing testbed. In *Proceedings of the 1988 ACM SIGMETRICS Conf. on measurement and Modeling of Computer Systems*, 1988.

[2] S.R. Goldschmidt and J.L. Hennessy. The accuracy of trace-driven simulations of multiprocessors. In *Proceedings of the 1993 ACM SIGMETRICS Conference on Measuring and Modeling of Computer Systems*, 1993.

[3] J. R. Larus and E. Schnarr. Eel: Machine-independent executable editing. In *SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 1995.

[4] J. Pierce and T. Mudge. Idtrace - a tracing tool for i486 simulation. Technical report, University of Michigan technical report CSE-TR-203-94, 1994.

[5] M. D. Smith. Tracing with pixie. Technical report, Technical Report, Stanford University, 1991.

[6] SPEC. *SPEC Newsletter*, vol. 7, issue 3, 1995.

[7] A. Srivastava and A. Eustace. Atom: A system for building customized program analysis tools. In *Proceedings of the SIGPLAN'94 Conference on Programming Language Design and Implementation*, 1994.

[8] C. Stephens, B. Cogswell, J. Heinlein, G. Palmer, and J. Shen. Instruction level profiling and evaluation of the ibm rs/6000. In *Proceeding of the 18th Annual International Symposium on Computer Architecture*, 1991.

[9] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The splash-2 programs: Characterization and methodological considerations. In *Proceedings of the 22thAnnual International Symposium on Computer Architecture*, 1995.