Xprofiler—Parallel Profiling Tool

By Jhy-Chun Wang, Xianneng Shen, and Ted Hoover

This is the first article in a series about the application development environment (Parallel Environment) of the IBM RS/6000 SP system. This article introduces Xprofiler, a profiling tool based on X-Windows that provides profiling at both a procedure and a source statement level. It combines graphical and textual displays of function call trees to reveal an application's performance profiling data.

Profiling the performance of an application normally involves counting the CPU time used by each procedure in the application and the number of times each procedure was called by other procedures within the application. Profiling is the first step in understanding an application's performance behavior: which procedures used the bulk of the CPU time or which procedures were called the greatest number of times. Once you identify these heavyweight procedures, you can focus your efforts on those procedures to improve the application's overall performance.

Xprofiler Characteristics

Xprofiler, a profiling tool based on X-Windows, has the capability of profiling at both procedure level and source statement level. Procedure-level profiling reveals the CPU time consumed by each procedure and the number of times each procedure was called by other procedures in the application. Source statement level profiling reveals the CPU time consumed by each source statement in a procedure. The procedure has to be compiled with the `-g` option in order to map profiling data to source statement level.

Xprofiler's graphical function call tree shows the CPU usage of each function within the application. The nodes in the tree represent program functions. The width of the node is proportional to the CPU time spent in the function and its descendant functions. The height of the node is proportional to the CPU time spent in the function itself.

Using this presentation scheme, a function spending little CPU time in its descendant functions will have a square shape; a node spending a significant amount of CPU time in its descendant functions (for example, the `main()` in C programs) will have a rectangular shape. In the graphical display, the larger a node, the more CPU time was consumed by the function (and/or its descendant functions). Xprofiler provides a link between the call tree structure and the source statements. The source code associated with a node in the call tree can be located using point-and-click.

The graphical user interface of the Xprofiler enables you to navigate between functions of the application and the system calls used in your application. You can zoom in and out of the main graphical display,

placing functions into different groups according to their location in files, and so on. This can simplify the main display or reveal the execution relationship of functions in different object files.

Xprofiler handles textual profiling reports well and provides statistical reports for parallel applications, such as the average time spent performing a function, tasks with minimum and maximum time in a function, and so on. Xprofiler will work on both the RS/6000 SP™ and the stand-alone RS/6000® server. It is packaged in the SP Parallel Environment product and currently available only within the parallel environment on the RS/6000 SP.

Xprofiler is similar to the AIX® `gprof` command in analyzing the profiling data and merging multiple output profiling files of a parallel application. Xprofiler can analyze an application's performance profiling data not only at the procedure level like `gprof`, but also at the source statement level. The `gprof` command presents the profiling information via plain text reports, which are not intuitive and often difficult to analyze if you use many functions/processors in an application. Xprofiler has a comprehensive graphical user interface that provides flexibility in navigating the profiling data.

Xprofiler works for both serial and parallel applications. When compiled with the proper compiler options, a sequential application generates only one profile data file (that is, the `gmon.out` file), while a parallel application will generate one profile data file for each task in the application. Once the profile data files are collected, Xprofiler can analyze the result. Xprofiler requires AIX 4.2.1 or higher to run on the system.

How to Use Xprofiler

To use Xprofiler, compile and link an application with the `-pg` compiler option, which is required for either sequential or parallel applications.[1]

The application generates the CPU usage and call count information when it executes and writes the data into one or more `gmon.out` files.

Xprofiler Features

Xprofiler not only supports all the functions and command-line options provided by `gprof`, but it also includes many functions that make analyzing an application's performance effective and seamless.

In addition to `gprof` functions, Xprofiler provides the following functions to help users navigate an application's performance profile data:

- Function call tree graphical display: Using functions as nodes and the caller-callee relationship as arcs, Xprofiler constructs a function call tree from an application's `gmon.out` files to represent the application's execution structure.

> ***Xprofiler can analyze an application's performance profiling data not only at the procedure level, but also at the source statement level.***

- Configuration file: With the configuration file, users can save one (customized) function call tree structure to be used in future Xprofiler sessions.
- Alter file search path: Users can specify the paths and search order in which a source code file or a library object file will be located.
- Overview window: This highlight area makes it easy to manipulate the graphical display.
- Summary mode/average mode: This is available when more than one `gmon.out` file is entered. This feature uses a different node presentation scheme in average mode to reveal

[1] *IBM Parallel Environment for AIX: Operation and Use,* Volume 2, Tools Reference. SC28-1980. Page 94.

```
        // The application must be compiled with the -pg option.
        // -g is required for source statement profiling. It is optional.
        // -O is required for compile optimization. It is optional.
xlC -O -g -pg -o xprof_1 xprof_1.C

        // Run the application. A gmon.out file will be generated.
xprof_1

        // bring up Xprofiler to analyze the profiling data.
xprofiler xprof_1 gmon.out
```

Figure 1. Profiling an application with Xprofiler

the load-balancing problem in a parallel application.

- Reconstruct function call tree: Users can remove nodes and/or arcs from the function call tree or add previously removed nodes/arcs back into the tree to customize or simplify the graphical display to focus on a particular area, such as the functions on a CPU usage critical path.
- Group functions into load units: Functions can be grouped into load units (that is, object files) to reveal the execution relationship between units and to simplify the graphical display.

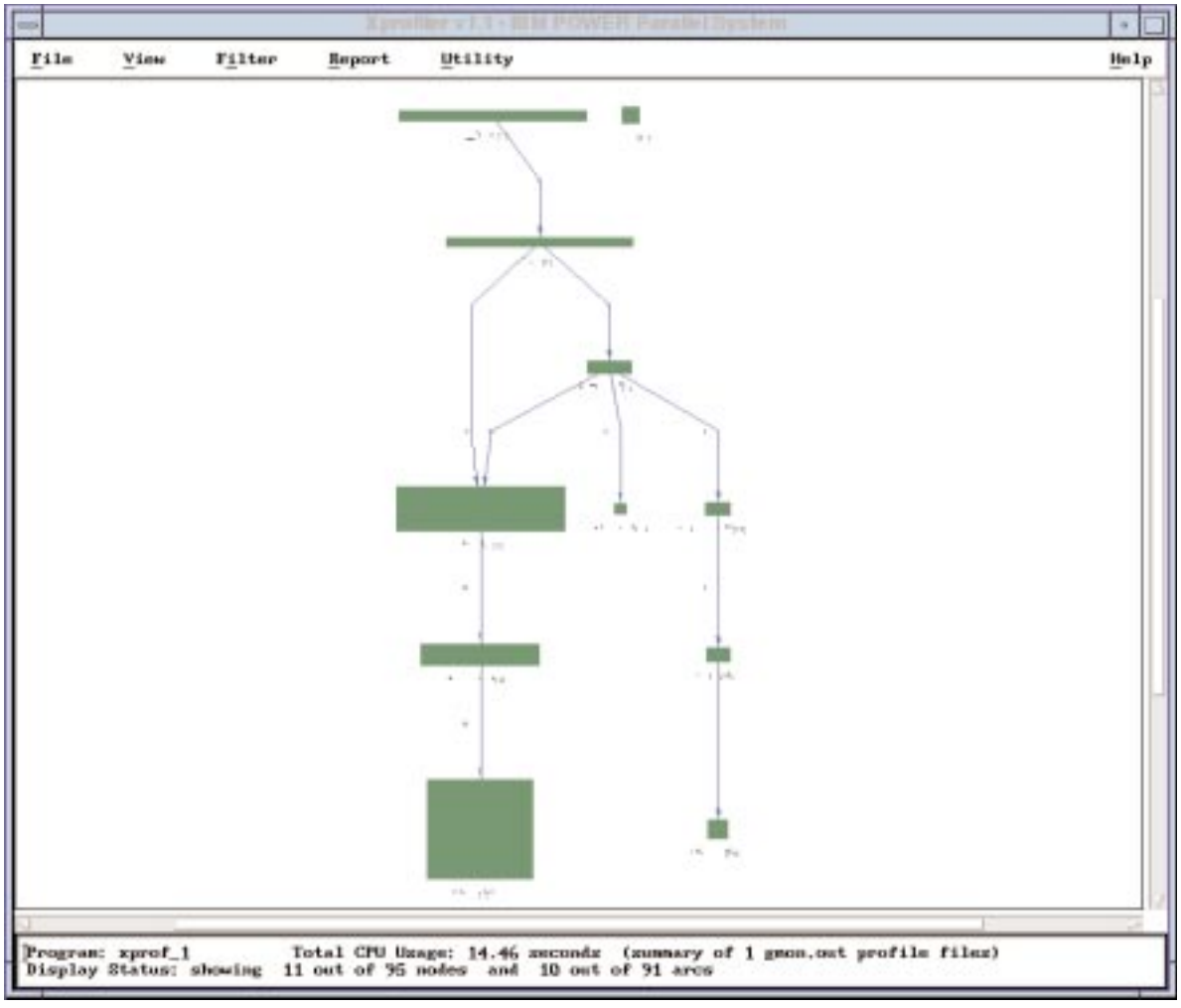


Figure 2. Xprofiler main graphical display

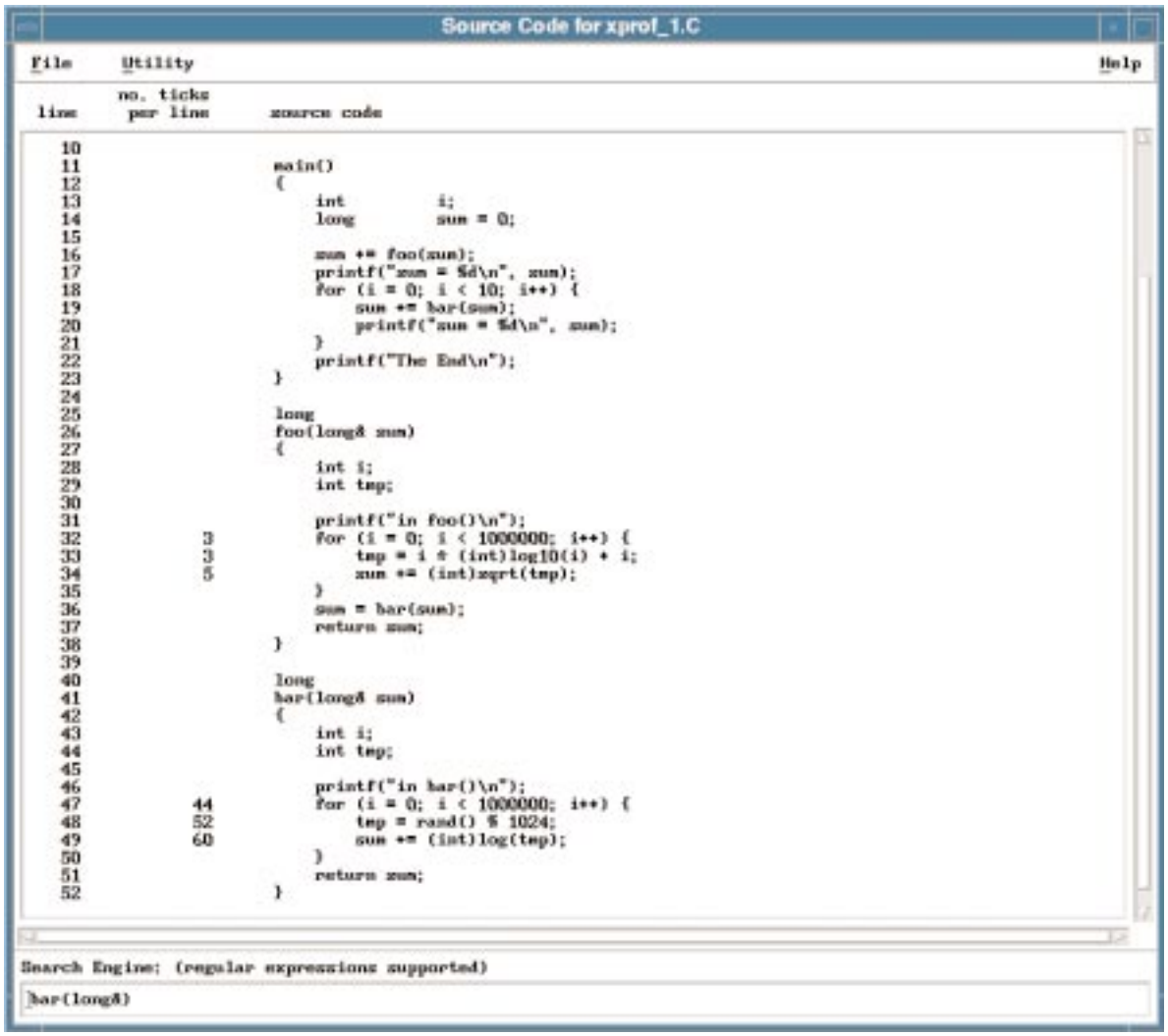


Figure 3. Xprofiler source code display

- Filter functions: The function call tree can be filtered by function names, CPU usage, or function call count information. These filter functions make it easier to reconstruct a function call tree.
- Textual reports: In addition to the `gprof` profiling reports, Xprofiler provides reports on the call count summary and load units' statistics data.
- Source code display: Profiling data (that is, CPU usage) can be mapped to source code by a point-and-click on the graphical display.
- Disassembler code display: Profiling data (CPU usage) can be mapped to disassembler code by a point-and-click on the graphical display.

Xprofiler supports computer languages such as Fortran 77, Fortran 90, High Performance Fortran, C, and C++.

Xprofiler Examples

Figure 1 shows an example program with three functions: `main()`, `foo()`, and `bar()`. If the source code is in the `xprof_1.C` file, execute the steps in Figure 1 to analyze the application using Xprofiler.

Figure 2 shows a simplified function call tree of the application `xprof_1`, where many system calls were removed from the display. The display employs a presentation principle whereby the larger a node is,

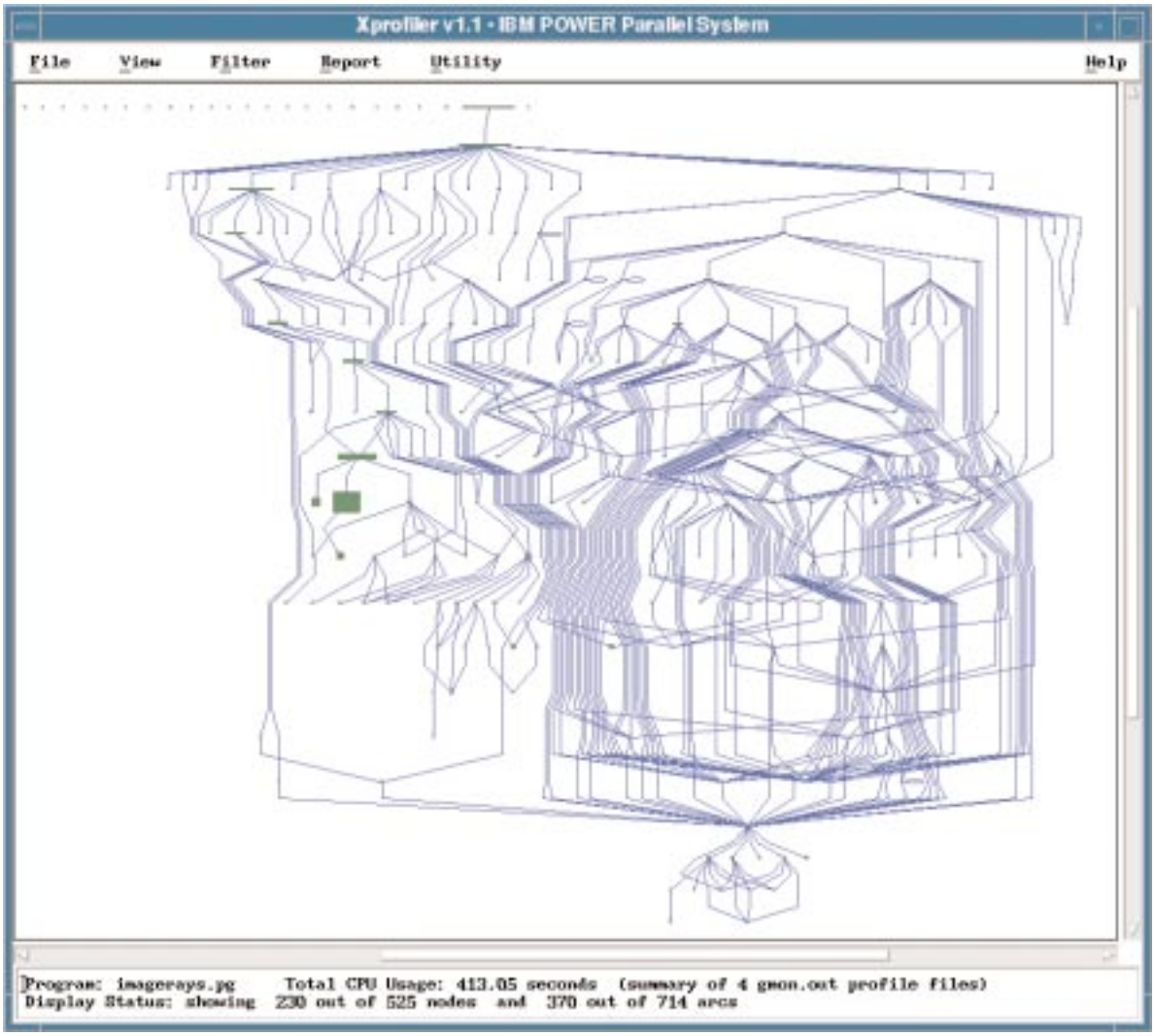


Figure 4. Xprofiler main graphical display of a large application

the more CPU time is spent by the node (remembering that each node represents a function in the application, including the system calls).

In this application, `function bar()` (at the bottom left area) used most of the CPU time. By clicking on the `node bar()` and selecting the source code display, the source code file that contains `function bar()` will be brought into the source code display.

Figure 3 shows the source code display of the file `xprof_1.C`, the second column (the number-of-ticks-per-line) in the display represents the number of ticks (on AIX, one tick equals 10 milliseconds) that occurred at each source statement. The statements that consumed more CPU time will have larger numbers of ticks mapped to them.

In Figure 3, source code lines 47 to 49 consumed 93% (156 ticks out of the total of 167 ticks) of the application's CPU usage; that is, they are the hot spot of the application. By combining Figures 2 and 3, you can easily locate the function that spent the most CPU time, then click on the node to bring up the function's source code file. The number of ticks mapped to each source statement will reveal where the bulk of CPU time was spent in a function. Once a CPU hot spot is located, you can devote some effort in that area to improve the application's performance.

Figure 4 shows the partial function call tree of a large application, which uses 525 functions, including system calls. Although the display seems complicated, a large node

on the center left area clearly stands out. It represents a function that consumed a lot of CPU time, and is therefore a good candidate for performance tuning.

Conclusion

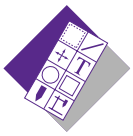
Performance profiling is an important step in performance tuning. Profiling data can reveal functions in an application that consume large amounts of CPU usage or call counts.

Xprofiler is superior to traditional command-line profile tools like `gprof`. It not only provides all the functions that `gprof` supports, but it also combines a graphical display with a source code browser and textual reports to provide a user-friendly environment for performance profiling on both sequential and parallel applications.

References

IBM Parallel Environment for AIX: Hitchhiker's Guide. GC23-3859.

IBM Parallel Environment for AIX: Operation and Use, Volume 2, Tools Reference. SC28-1980.



***Jhy-Chun Wang**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: wangjc@us.ibm.com. Dr. Wang is an advisory software engineer in the RS/6000 SP Group. His current interests include performance analysis and portable high-performance computing tools. Dr. Wang holds a PhD in Computer Science from Syracuse University.*

***Xianneng Shen**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: xshen@us.ibm.com. Dr. Shen is a programming consultant in the RS/6000 SP Teraplex Integration Center. His current interests include scalable parallel algorithms, scalable I/O, parallel database, data warehouse, and data mining implementations. Dr. Shen holds a PhD in Electrical Engineering from Syracuse University and an MS in Computer Engineering from Syracuse University.*

***Ted Hoover**, IBM Corporation, 522 South Road, Poughkeepsie, NY 12601. E-mail: hoov@us.ibm.com. Mr. Hoover is an advisory software engineer and the team lead for the Application Development Tools team in the RS/6000 SP Group. He has an MS in Computer Science from Rensselaer Polytechnic Institute and a BS in Computer Science from the University of Pittsburgh at Johnstown.*