Introducing Debugging and Profiling for Parallel Programs

Dr. J. Lakshmi
SERC, Indian Institute of Science
Bengaluru-12

TALK OUTLINE

Why debuggers?

What can they do to help you enhance your program development?

Parallel program debugging

What are profilers and why you could need them?

PROGRAM DEBUGGING

Why do we need debuggers?

- Programming errors not detectable by compilation or linking
- Such errors cause change in runtime behavior

WHAT IS A DEBUGGER?

“A software tool that is used to detect the source of program or script errors, by performing step-by-step execution of application code and viewing the content of code variables.”

-MSDN

WHAT IS A DEBUGGER? (CON'T)

A debugger is *not an IDE*

- Though the two can be integrated, they are separate entities.

A debugger loads in a program (compiled executable, or interpreted source code) and allows the user to trace through the execution.

Debuggers typically can do disassembly, stack traces, variable and expression watches, and more.

OTHER FORMS OF DEBUGGING

Periodic printf/cout/print/write … etc.

- Statements with relevant information

Assert statements

Desk Checking or Code Walkthroughs!

WHY USE A DEBUGGER?

No need for precognition of what the error might be.

Flexible

- Allows for “live” error checking – no need to re-write and re-compile when you realize a certain type of error may be occurring
- Dynamic
- Can view the entire relevant scope

RELUCTANCE TO *USING A DEBUGGER*

With simple errors, may not want to bother with starting up the debugger environment.

- Obvious error
- Simple to check using prints/asserts

Hard-to-use debugger environment

Error occurs in optimized code

Changes execution of program (error doesn't occur while running debugger)

DEBUGGERS FOR COMPILED LANGUAGES

Debuggers are special programs that can

- Read your executables and connect with the source code
- Maintain runtime order, scope and variables of your program as it is being executed
- Generally, would like information about source code (not normally included in compiled executables)
- Work on a lower level

Need special "debug" executables.

FUNCTIONS OF A DEBUGGER

- Disassembly
- Execution Tracing/Stack tracing
- Symbol watches

DISASSEMBLY

- Most basic form of debugging
- Translating machine code into assembly instructions that are more easily understood by the user.
- Typically implementable as a simple lookup table
- No higher-level information (variable names, etc.)

EXECUTION TRACING

- Follows the program through the execution. Users can step through line-by-line, or use breakpoints.
- Typically allows for “watches” on – registers, memory locations, symbols
- Allows for tracing up the stack of runtime errors (back traces)
- Allows user to trace the causes of unexpected behavior and fix them

SYMBOL INFORMATION

- Problem – a compiler/assembler translates variable names and other symbols into internally consistent memory addresses
- How does a debugger know which location is denoted by a particular symbol?
- We need a “debug” executable.

DEBUG VS. RELEASE BUILDS

Debug builds usually are *not optimized*

Debug executables contain:

- program's symbol tables
- location of the source file
- line number tags for assembly instructions.

DEBUGGING PARALLEL PROGRAMS

Parallel programs introduce additional issues like deadlocks and race conditions

- Timing
- Synchronization

Side-effects

- Error behavior may not be repeatable!
- Error location too may change in different runs!

Debugging Parallel Programs

TIMING YOUR CODE

```
/usr/bin/time –p a.out
    real 9.95 user 9.86 sys 0.06
```

You can also time a portion of your code by using clock() system call!

PROFILERS

What are profilers?

- Profilers are tools that help you analyze where your program spent its time or put its code in memory while in execution.

Time Profilers:

- Tells you where your program spent its time
- Tells you which functions called which other functions while it was executing

Space Profilers:

- Also called “heap profiling” or “memory profiling”
- Space profiling is useful to help you reduce the amount of memory your program uses.

HOW DO THEY WORK – TIME PROFILER?

Time profiler:

- Profiling works by changing how every function in your program is compiled so that when it is called, it will stash away some information about where it was called from.
- From this, the profiler can figure out what function called it, and can count how many times it was called

HOW DO THEY WORK – SPACE PROFILER?

Space Profiler:

- Stops execution and examines the stack
- Stops execution when a page of memory is allocated
- Collects Data about which function asked for the memory

HOW DO THEY WORK – PROFILED DATA?

- After the data is collected by the profiler, an interpreter must be run to display the data in an understandable format
- Can be text-based or graphical

WHY DO I NEED A TIME PROFILER?

Find where the program is spending most of it’s time

- That’s where you should focus optimization efforts

The program performs the proper functions, but is too slow

- Important in real time systems
- Important to web applications

The program is too large or too complex to analyze by reading the source

WHY DO I NEED A SPACE PROFILER?

- The program needs to use a fixed amount of memory
- The program is too large to conceive of the overall memory usage or how often memory requests are made
- Profilers can show the memory usage of libraries used by your program

SOME PROFILER EXAMPLES – GPROF

gprof – OpenSource Profiler (http://www.thegeekstuff.com/2012/08/gprof-tutorial/)

- compile programs with the –pg option
- execute program to generate data
- run **gprof** to interpret the profiled data

GPROF SAMPLE DATA – FLAT PROFILE

```
Each sample counts as 0.01 seconds.

 %   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
33.34     0.02     0.02      7208     0.00     0.00   open
16.67     0.03     0.01       244     0.04     0.12   offtime
16.67     0.04     0.01         8     1.25     1.25   memccpy
16.67     0.05     0.01         7     1.43     1.43   write
16.67     0.06     0.01                               mcount
 0.00     0.06     0.00       236     0.00     0.00   tzset
 0.00     0.06     0.00       192     0.00     0.00   tolower
```

GPROF SAMPLE DATA – CALL GRAPH

```
index % time    self  children    called     name
-----------------------------------------------
                0.00    0.05       1/1           main [2]
[3]    100.0    0.00    0.05       1         report [3]
                0.00    0.03       8/8           timelocal [6]
                0.00    0.01       1/1           print [9]
                0.00    0.01       9/9           fgets [12]
                0.00    0.00      12/34          strncmp <cycle 1> [40]
                0.00    0.00       8/8           lookup [20]
                0.00    0.00       1/1           fopen [21]
                0.00    0.00       8/8           chewtime [24]
                0.00    0.00       8/16          skipspace [44]
-----------------------------------------------
```

SOME PROFILER EXAMPLES – SPACE

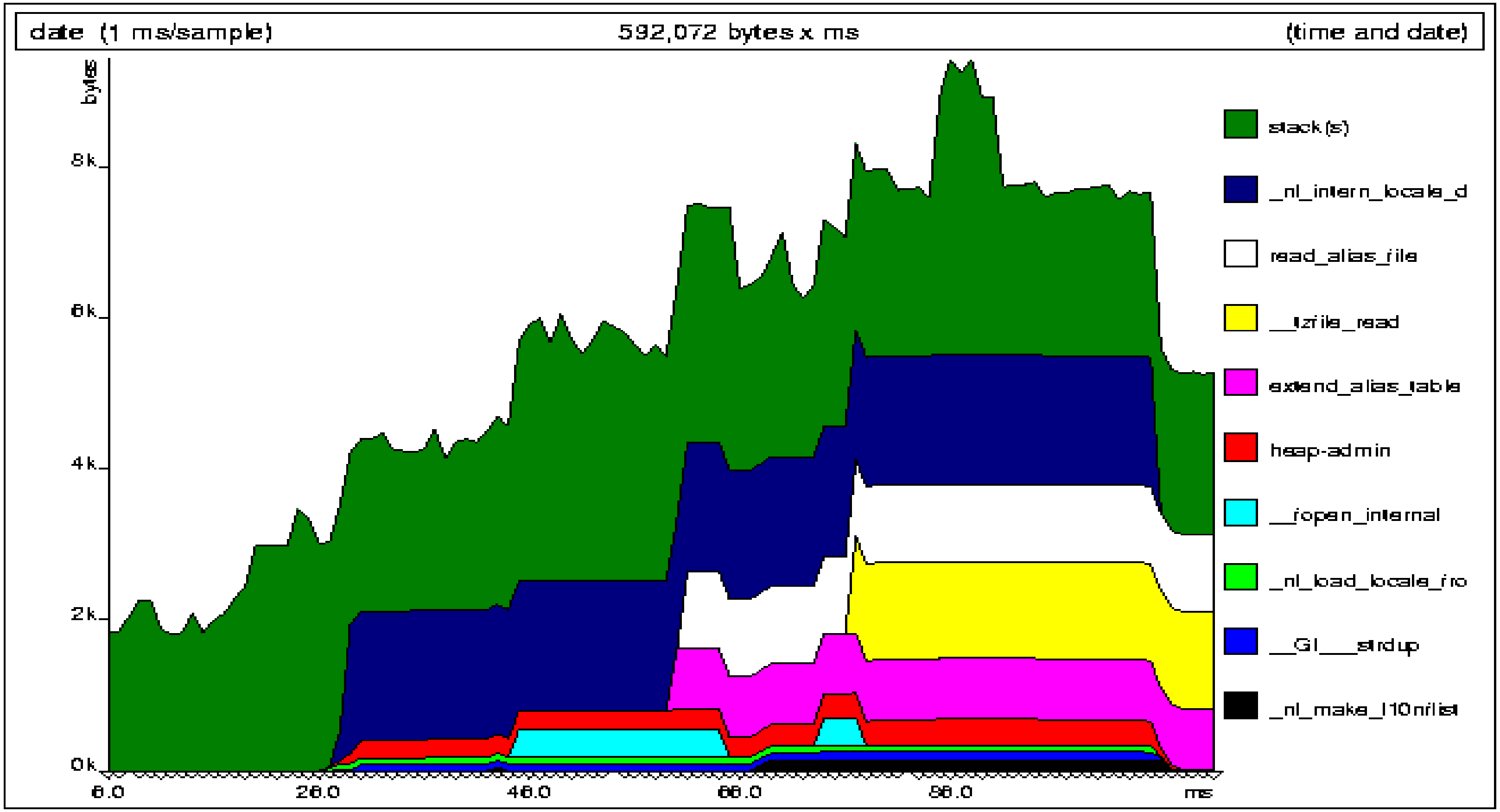
Massif (http://valgrind.org/docs/manual/ms-manual.html)

- Space Profiler for C and C++
- Provides relative space data on 5 different areas:
  - Heap blocks
  - Heap administration blocks
  - Stack sizes
  - Code size
  - Data size

MASSIF SAMPLE DATA - BASIC

```
==1012== Total spacetime:   917,098,589 ms.B
==1012== heap:              0.0%
==1012== heap admin:             0.0%
==1012== stack(s):           0.0%
==1012== static code:        44.4%
==1012== static data:       55.3%
```

MASSIF SAMPLE DATA – SPACE-TIME GRAPH



READING LIST – DEBUGGING & PROFILING PARALLEL CODES

Debugging and Profiling basics (https://cvw.cac.cornell.edu/Profiling/debugging_distributed)

Profiling and optimizing serial and parallel codes (https://portal.tacc.utexas.edu/c/document_library/get_file?uuid=fc609b77-b727-4bff-81a4-d30caa4013d4&groupId=13601)

Identifying bottlenecks in parallel codes (http://www.it.northwestern.edu/bin/docs/research/bottlenecks-in-HPC.pdf)

THANKYOU!

jlakshmi@iisc.ac.in