CLAIMS

1. A concurrency-enhanced system, comprising:

an execution stack of a program, the execution stack including execution frames over a time period of interest;

a memory, the memory configured by the execution stack, the memory also configured by behavior-driven stack scan optimization (BDSSO) software;

a processor in operable communication with the memory, the processor configured to execute the BDSSO software to perform BDSSO steps which include (a) obtaining execution stack frame occurrence data, (b) determining from the execution stack frame occurrence data, for each of a plurality of execution frames, a respective frame execution likelihood, (c) selecting a stack scan depth based at least in part on the frame execution likelihoods, the selected stack scan depth being less than a full depth of the entire execution stack, (d) installing a garbage collection scan return barrier at the selected stack scan depth, and then (e) allowing a tracing garbage collector to scan the execution stack below the scan return barrier while the program is running;

whereby system concurrency is increased by performing the BDSSO steps, because the program runs concurrently with the garbage collector while the garbage collector scans the execution stack below the scan return barrier unless the program executes a frame return operation that hits the scan return barrier.

2. The system of claim 1, further comprising execution samples created by a profiler tool during an execution of the program, and wherein the BDSSO software obtains the execution stack frame occurrence data at least in part by analyzing the execution samples.

3. The system of claim 1, further comprising a managed runtime, and wherein the BDSSO software obtains the execution stack frame occurrence data at least in part by gathering stack data at garbage collection safepoints utilizing the managed runtime.

4. The system of claim 1, wherein the system is further characterized by differing from a fixed depth stack scanning system in at least one of the following ways:

the garbage collection scan return barrier is installed below at least two frames of the execution stack;

the system has a plurality of programs, and the selected stack scan depth varies between programs; or

the selected stack scan depth varies for different time periods of interest for a given program during an execution of the given program.

5. The system of claim 1, wherein the processor comprises N cores with N being at least two, the program comprises M threads with 2 <= M <=N, each

thread has a respective execution stack, and the BDSSO software performs the BDSSO steps on a per-thread basis for each of the M threads.

6. The system of claim 1, wherein the execution stack frame occurrence data includes return addresses and is free of each of the following: data stating an elapsed execution time, and data stating a processor cycles count.

7. A concurrency enhancement method for optimizing execution stack tracing for garbage collection, the method comprising:

obtaining execution stack frame occurrence data for an execution stack;

automatically determining from the execution stack frame occurrence data, for each of a plurality of execution frames, a respective frame execution likelihood;

automatically selecting a stack scan depth based at least in part on the frame execution likelihoods, the selected stack scan depth being less than a full depth of the entire execution stack;

automatically installing a garbage collection scan return barrier at the selected stack scan depth; and

allowing the tracing garbage collector to scan the execution stack only to the selected stack scan depth;

whereby concurrency is increased by performing the method, by an execution of a garbage collector stack scanning software concurrently with an execution of a mutator software, in comparison to scanning the execution stack for garbage collection without performing the method.

8. The method of claim 7, wherein the execution stack frame occurrence data includes stack frame snapshots for different points during execution of a program, and determining a frame execution likelihood includes tallying how many times a particular set of execution frames occurs in a set of stack frame snapshots.

9. The method of claim 7, wherein determining a frame execution likelihood includes at least one of the following: comparing an elapsed execution time for a partial execution of a routine to an estimated execution time for a complete execution of the routine, or determining an estimated completion time for completing execution of the routine.

10. The method of claim 7, wherein obtaining execution stack frame occurrence data comprises periodically sampling the execution stack during execution of a program.

11. The method of claim 7, wherein the execution stack frame occurrence data includes stack frame snapshots for different points during execution of a program, a majority of the snapshots each show a stack depth of at least one hundred frames, and the selected stack scan depth is less than ten and more than one.

12. The method of claim 7, wherein determining the frame execution likelihood and selecting the stack scan depth is performed without reading or otherwise relying on any of the following: data stating an elapsed execution time for

~~any stack frame, data stating a processor cycles count for any stack frame, or data stating a memory consumption amount for any stack frame.~~

~~13. The method of claim 7, wherein obtaining execution stack frame occurrence data comprises at least one of the following:~~

~~obtaining execution samples created by a profiler tool; or~~

~~utilizing a managed runtime for profiling at garbage collection safepoints.~~

~~14. The method of claim 7, wherein the mutator software hits the scan return barrier, and the method further comprises: removing the scan return barrier, automatically selecting a different stack scan depth, and automatically installing the scan return barrier at the different stack scan depth.~~

~~15. The method of claim 7, further characterized in at least one of the following ways:~~

~~the mutator software includes an application program, and the method comprises performing at least the obtaining, determining, selecting, and installing with each of a plurality of execution stacks of respective threads on a per-thread basis for at least two threads of the application program;~~

~~the execution stack corresponds to a thread of the program, the method repeats the selecting and installing with different stack scan depths at different times during execution of the thread, and each time the garbage collection scan return barrier is installed below at least two frames of the execution stack; or~~

~~the execution stack frame occurrence data includes stack frame snapshots for different points during execution of the program, at least 80% of the snapshots each show a stack depth of at least eighty frames, and the selected stack scan depth is less than nine.~~

1. A concurrency-enhanced system, comprising:

an execution stack of a program, the execution stack including execution frames over a time period of interest;

a memory, the memory configured by the execution stack, the memory also configured by behavior-driven stack scan optimization (BDSSO) software;

a processor in operable communication with the memory, the processor configured to execute the BDSSO software to perform BDSSO steps which include (a) obtaining execution stack frame occurrence data, (b) determining from the execution stack frame occurrence data, for each of a plurality of execution frames, a respective frame execution likelihood indicating a computed likelihood that a respective execution frame will be accessed while a tracing garbage collector scans the execution stack, (c) selecting a stack scan depth based at least in part on the frame execution likelihoods, the selected stack scan depth being less than a full depth of the entire execution stack, (d) installing a garbage collection scan return barrier at the selected stack scan depth, and then (e) allowing the tracing garbage collector to scan the execution stack below the scan return barrier while the program is running.

2. The system of claim 1, further comprising execution samples created by a profiler tool during an execution of the program, and wherein the BDSSO software obtains the execution stack frame occurrence data at least in part by analyzing the execution samples.

3. The system of claim 1, further comprising a managed runtime, and wherein the BDSSO software obtains the execution stack frame occurrence data at least in part by gathering stack data at garbage collection safepoints utilizing the managed runtime.

4. The system of claim 1, wherein the system is further characterized by differing from a fixed depth stack scanning system in at least one of the following ways:

the garbage collection scan return barrier is installed below at least two frames of the execution stack;

the system has a plurality of programs, and the selected stack scan depth varies between programs; or

the selected stack scan depth varies for different time periods of interest for a given program during an execution of the given program.

5. The system of claim 1, wherein the processor comprises N cores with N being at least two, the program comprises M threads, wherein M is greater than or equal to 2 but less than or equal to N, each thread has a respective execution stack, and the BDSSO software performs the BDSSO steps on a per-thread basis for each of the M threads.

6. The system of claim 1, wherein the execution stack frame occurrence data includes return addresses and is free of each of the following: data stating an elapsed execution time, and data stating a processor cycles count.

7. A concurrency enhancement method for optimizing execution stack tracing for garbage collection, the method comprising:

obtaining execution stack frame occurrence data for an execution stack;

automatically determining from the execution stack frame occurrence data, for each of a plurality of execution frames, a respective frame execution likelihood indicating a computed likelihood that a respective execution frame will be accessed while a tracing garbage collector scans the execution stack;

automatically selecting a stack scan depth based at least in part on the frame execution likelihoods, the selected stack scan depth being less than a full depth of the entire execution stack;

automatically installing a garbage collection scan return barrier at the selected stack scan depth; and

allowing the tracing garbage collector to scan the execution stack only to the selected stack scan depth.

8. The method of claim 7, wherein the execution stack frame occurrence data includes stack frame snapshots for different points during execution of a program, and determining a frame execution likelihood includes tallying how many times a particular set of execution frames occurs in a set of stack frame snapshots.

9. The method of claim 7, wherein determining a frame execution likelihood includes at least one of the following: comparing an elapsed execution time for a partial execution of a routine to an estimated execution time for a complete execution of the routine, or determining an estimated completion time for completing execution of the routine.

10. The method of claim 7, wherein obtaining execution stack frame occurrence data comprises periodically sampling the execution stack during execution of a program.

11. The method of claim 7, wherein the execution stack frame occurrence data includes stack frame snapshots for different points during execution of a program, a majority of the snapshots each show a stack depth of at least one hundred frames, and the selected stack scan depth is less than ten and more than one.

12. The method of claim 7, wherein determining the frame execution likelihood and selecting the stack scan depth is performed without reading or otherwise relying on any of the following: data stating an elapsed execution time for any stack frame, data stating a processor cycles count for any stack frame, or data stating a memory consumption amount for any stack frame.

13. The method of claim 7, wherein obtaining execution stack frame occurrence data comprises obtaining execution samples created by a profiler tool.

14. The method of claim 7, wherein obtaining execution stack frame occurrence data comprises utilizing a managed runtime for profiling at garbage collection safepoints.

15. The method of claim 7, wherein the method comprises performing at least the obtaining, determining, selecting, and installing with each of a plurality of execution stacks of respective threads on a per-thread basis for at least two threads of an application program.

16. A computer-readable storage medium configured with data and instructions which upon execution by a processor cause a computing system to perform a concurrency enhancement method of optimizing execution stack tracing for garbage collection, the method comprising:

obtaining execution stack frame occurrence data for an execution stack of a program;

automatically determining from the execution stack frame occurrence data, for each of a plurality of execution frames, a respective frame execution likelihood indicating a computed likelihood that a respective execution frame will be accessed while a tracing garbage collector scans the execution stack;

automatically selecting a stack scan depth based at least in part on the frame execution likelihoods, the selected stack scan depth being less than a full depth of the entire execution stack;

automatically installing a garbage collection scan return barrier at the selected stack scan depth; and

allowing the tracing garbage collector to scan the execution stack only to the selected stack scan depth.

17. The computer-readable storage medium of claim 16, wherein the program hits the scan return barrier, and the method further comprises: removing the scan return barrier, automatically selecting a different stack scan depth, and automatically installing the scan return barrier at the different stack scan depth.

18. The computer-readable storage medium of claim 16, wherein the execution stack corresponds to a thread of the program, the method repeats the selecting and installing with different stack scan depths at different times during execution of the thread, and each time the garbage collection scan return barrier is installed below at least two frames of the execution stack.

19. The computer-readable storage medium of claim 18, wherein the execution stack frame occurrence data includes stack frame snapshots for different points during execution of the program, at least 80% of the snapshots each show a stack depth of at least eighty frames, and the selected stack scan depth is less than nine.

20. The computer-readable storage medium of claim 16, wherein determining a frame execution likelihood includes comparing a frame execution probability to a frame execution probability threshold.