Lab 3: Finding Parallelism Issues - Linux

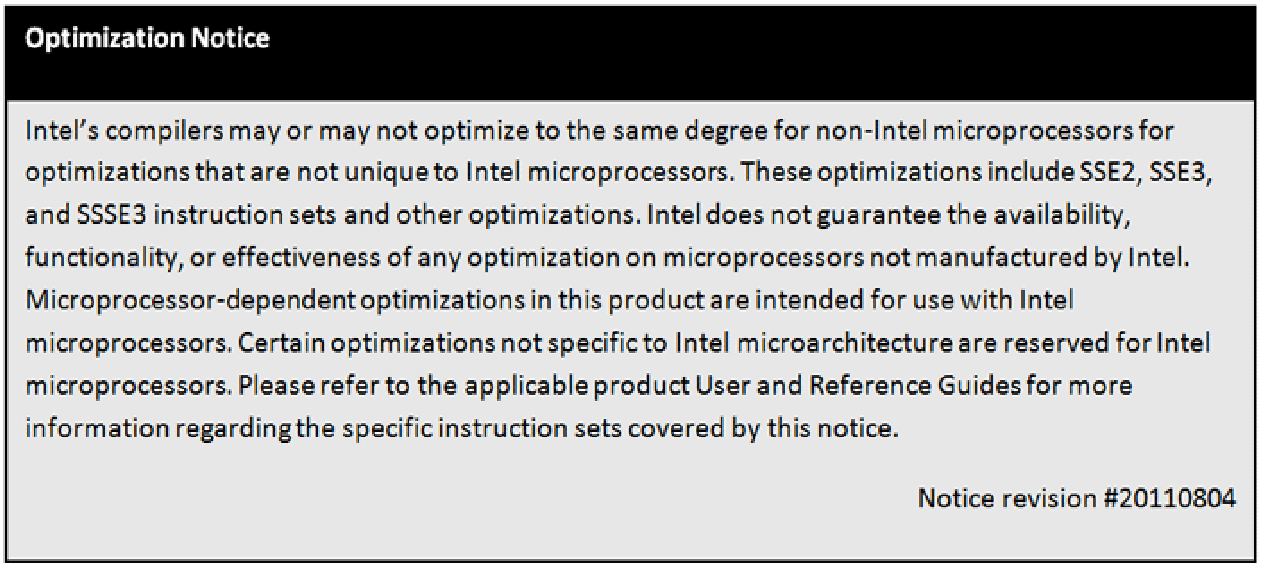
Developer Product Division

Developer Product Division
Software and Service Group

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.



Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Table of Contents

Lab 3: Finding Parallelism Issues

Time Required	Thirty minutes
Objective	In this lab session, you will use Intel® VTune™ Amplifier XE to find parallelism issues in an application. After successfully completing this lab's activities, you will be able to: • Use the Intel® Vtune™ Amplifier XE to find parallelism issues caused by synchronization locks or other waits.

Activity 1 – Collect Locks-And-Waits data

Time Required	Ten minutes
Objective	• Run the application while collecting parallelism data

1. Click on the "New Analysis" button.
2. Select "Algorithm Analysis->Locks-And-Waits".
3. Click the "Start" button on the right side of the window - The tachyon application will run. Note that as the application runs it draws an image of several different silver balls on the screen. Make a note of the execution time displayed in the application window as before.
4. After the application completes the Intel® VTune™ Amplifier XE will spend some time analyzing the data. When it is finished analyzing, the Hotspots pane appears. Note the analysis explanation pane comes up. Read it and then clear the pane.

   At this point the application has run to completion and the Intel® VTune™ Amplifier XE 2016 displays the analyzed results.

Review Questions

Question 1: What is the result screen that appears after clearing the analysis explanation pane?

Question 2: What useful data is in this first screen?

Activity 2 – Find Causes of Poor Parallelism

Time Required	Ten minutes
Objective	• Use the Intel® VTune™ Amplifier XE to find a cause of poor parallelism
Codes Description	• Tachyon is a 2-D raytracer/rendering program that displays an image

1. Make sure the "Bottom-up" tab is selected. Notice at or near the top is a Sync Object labeled "Mutex 0xnnnnnnnn". This is a mutex that is in the user code and is limiting parallelism as shown by the large amount of wait time indicated and the amount of "Poor" time (fewer than the available number of CPUs used) shown.
2. Click on the triangle to the left of the term "Mutex" to expand the list of callers to that Mutex. Notice that the mutex is referenced by the function named draw_task.
3. Double click on the function name "draw_task". The source and assembly view are now displayed. Note that there is a mutex that is used in the draw_task function.

   You may want to make more of the source code visible by clicking and dragging the vertical line between the source and assembly panes to the right, and/or dragging the horizontal bar above the timeline down.

   This mutex is actually not needed. The "for" loop is already thread safe. It was accidentally inserted as an extra precaution in a moment of panic ☺. This causes some unneeded serial, "poor CPU usage" time.
4. To improve the speed and parallelism of the application, exit the Amplifier XE GUI and use vi to comment out the calls to pthread_mutex_lock and pthread_mutex_unlock inside "draw_task", then rebuild. Finally, run the app and notice the execution time in the title bar when the application completes.
5. Rerun the Concurrency analysis as you did in Lab2 and see if the app is now faster. Also, see if the offending mutex is no longer on the list of sync objects.

Review Questions

Question 1: Does the parallelism look better in the Timeline at the bottom of the screen?

Question 2: How much faster is the application?