

# Getting Compiler Advice from the Optimization Reports

## Getting Started Guide

---

An optimizing compiler can do a lot better with just a few tips from you. We've integrated the Intel® compilers with Intel® VTune™ Performance Analyzer to make this easy.

The Intel compiler optimization and vectorization reports contain a wealth of information to make your application faster. VTune analyzer locates your critical, time consuming "hot spot" and filters the compiler optimization report to show only the lines that apply to the code selected. Now you can see what the compiler optimized and choose pragmas to further improve performance.

For example, a single click tells you that the compiler didn't optimize your critical loop because of an assumed vector dependency. You know there is no dependency and insert a pragma telling the compiler to ignore it which may result in a 20 percent improvement in run time.

Currently, compiler report filtering works exclusively with Intel® C++ and Fortran Compilers 9.1 and higher, but it utilizes a standard format open to other compilers.

## Contents

Disclaimer and Legal Information .....	2
1 Build the Application .....	2
2 Identify a Hotspot using the VTune™ Performance Analyzer .....	4
3 Viewing the Compiler Reports .....	7
4 Optimizing the Code .....	10



## Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

The software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

This document as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Developers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Improper use of reserved or undefined features or instructions may cause unpredictable behavior or failure in developer's software code when running on an Intel processor. Intel reserves these features or instructions for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from their unauthorized use.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, I960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\* Other names and brands may be claimed as the property of others.

Copyright © 2007, Intel Corporation.

# 1 *Build the Application*

---

Start with the application used in the final step of the Getting Started Guide of the VTune analyzer (see <installdir>vtune/doc/ Getting\_Started.pdf, if you have the analyzer installed). The first step is to build the Getting Started application, gsexample4c.c, using the Intel C++ compiler and aggressive optimizations.

**TIP:** If you do not have the Intel® Compiler installed, you can use the prebuilt application file included in this release.

After building the application, time it and use this as your baseline measurement.

1. Build the application with the Intel C++ compiler, version 9.1 using the following command:

```
icc -g -O3 -xP -o gsexample4c gsexample4c.c
```



2. Time the application using:

```
./gsexample4c datefile.txt
```

3. Record the “Average loop iterations/sec” on the test system, the value was 37,600.

The `-O3` option tells the compiler to perform high level optimizations and the `-xP` option tells the compiler to optimize for Intel® Core™ Duo, Core Solo, and Pentium 4 processors with Streaming SIMD Extensions 3 (see the Quick-Reference Guide to Optimization with Intel® Compilers).

You need to specify the following compiler options when you build the application with the Intel C++ compiler. These reports are later viewed from the VTune analyzer user interface.

Compiler Option	Description
<code>-opt-report</code>	Generates the optimization report
<code>-opt-report-level max</code>	Generates the optimization report with the maximum level of details.
<code>-opt-report-file gsexample4c.opt</code>	Outputs the report to the file <code>gsexample4c.opt</code>
<code>-vec-report3</code>	Generate the vectorization report with maximum details to stdout
<code>-fno-inline</code>	Disables inlining of functions. This setting simplifies identification of performance related optimizations

This is the compiler invocation:

```
icc -g -O3 -xP -Ob0 -opt-report -opt-report-level max -opt-report-file gsexample4c.opt -vec-report3 gsexample4c.c -o gsexample4c > gsexample4c.vec
```

The compiler builds the application and generates the following files:

- `gsexample4c.opt` optimization report
- `gsexample4c.vec` vectorization report

Combine these files using the following command


```
cat gsexample4c.opt gsexample4c.vec > gsexample4c.vor
```



## 2 Identify a Hotspot using the VTune™ Performance Analyzer

---

Next, use the VTune analyzer to collect sampling data and identify a “hotspot” in the application. A hotspot is a portion of code that contributes significantly to the total processing of the application.

1. Start the graphical user interface of the VTune analyzer using the `vtlec` command. If prompted for a workspace, create a new workspace for this example by typing in a unique, unused name, for example, `workspace_optreportexample`.
2. Click  Add Tuning Activity on the toolbar to open the **New** dialog box.
3. Click Tuning Activity in the right side of the dialog box and select **First Use Wizard** from the list of available wizards.
4. Set the following configuration options:
  - a. For **Application to launch**, use the **Browse...** button to locate the application: `gsexample4c`
  - b. For **Application arguments**, type in `datafile.txt`
  - c. For **Working directory**, specify the directory containing the application and data file
5. Click **Finish** to complete the configuration and begin collecting profiling data.

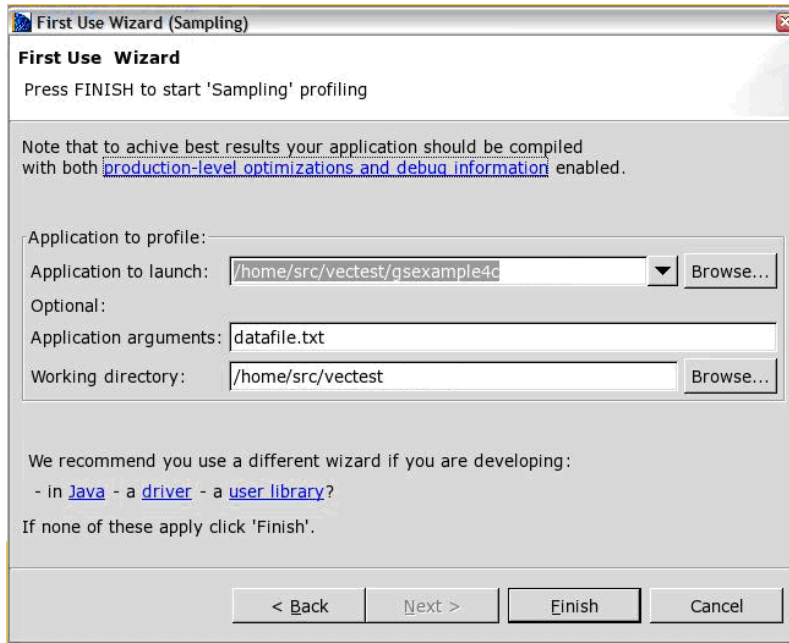


Figure 1: First Use wizard settings

The application terminates after 10 seconds and the VTune analyzer displays the results. At this point, the VTune analyzer may prompt for the kernel's location. This is because much of the activity of the application is now in the kernel's processing of the `_read()` function and an uncompressed kernel file was not located by the VTune analyzer. Select **Skip Always** since you will not be tuning any kernel code.

Next, the VTune analyzer displays the **Sampling Results** window with the **Most Active Function In Your Application** table. The function `ProcessBuffer()` is consuming most of the cycles in this application. This application is the most optimized version of the sample application used in the Getting Started Guide.

When prompted for kernel location:

Select **Skip** if you intend to tune the kernel in this project.

Select **Skip Always** if you do not intend to tune

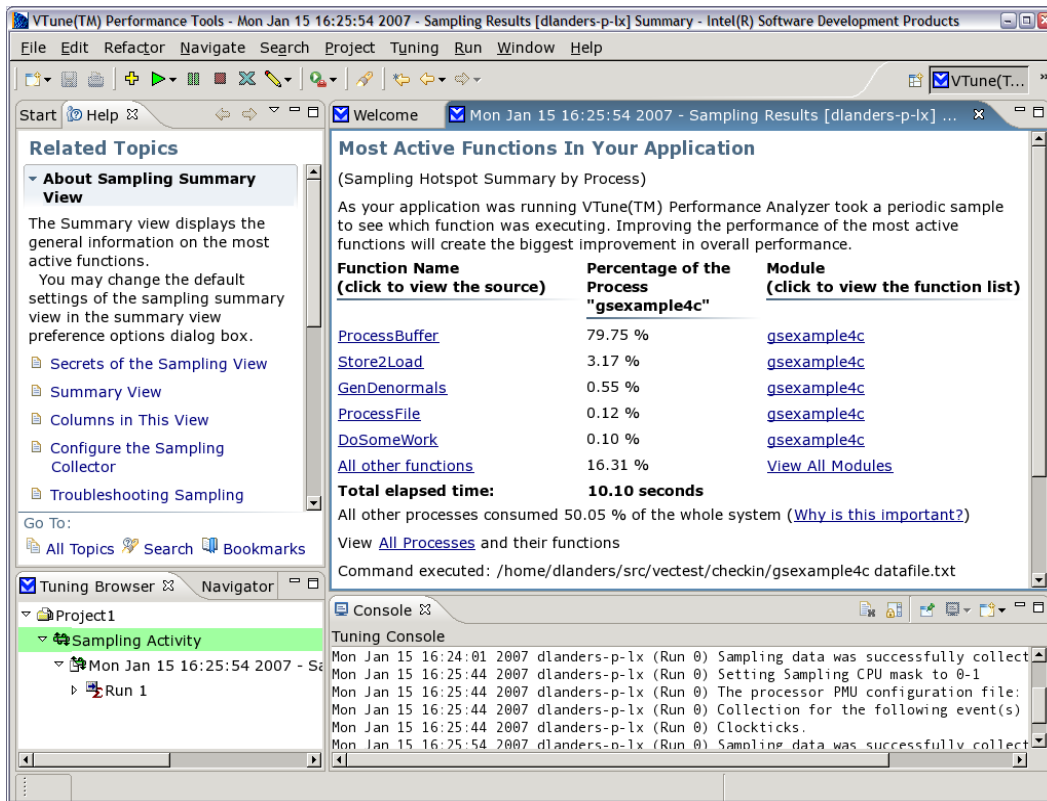


Figure 2: First Use Activity results

Click the first function name, **ProcessBuffer** to open the source view for this function. The source view automatically scrolls to the beginning of this function.

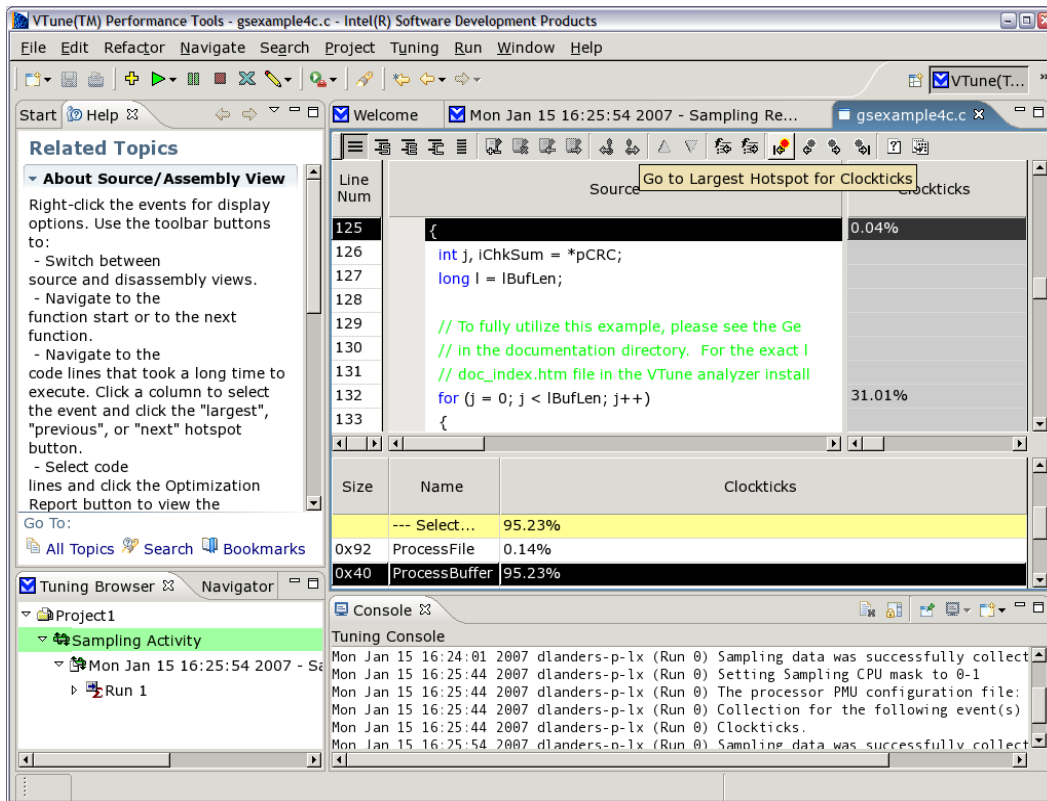




Figure 3: Source View for ProcessBuffer

Next, click Go to Largest Hotspot for Clockticks  to scroll to the line with the most Clockticks events. Notice that it is in the loop of the `ProcessBuffer()` function.

## 3 Viewing the Compiler Reports

To view the opt report for this code, select the source lines that comprise the 'for' loop and press the Optimization Report button  on the right of the source view toolbar

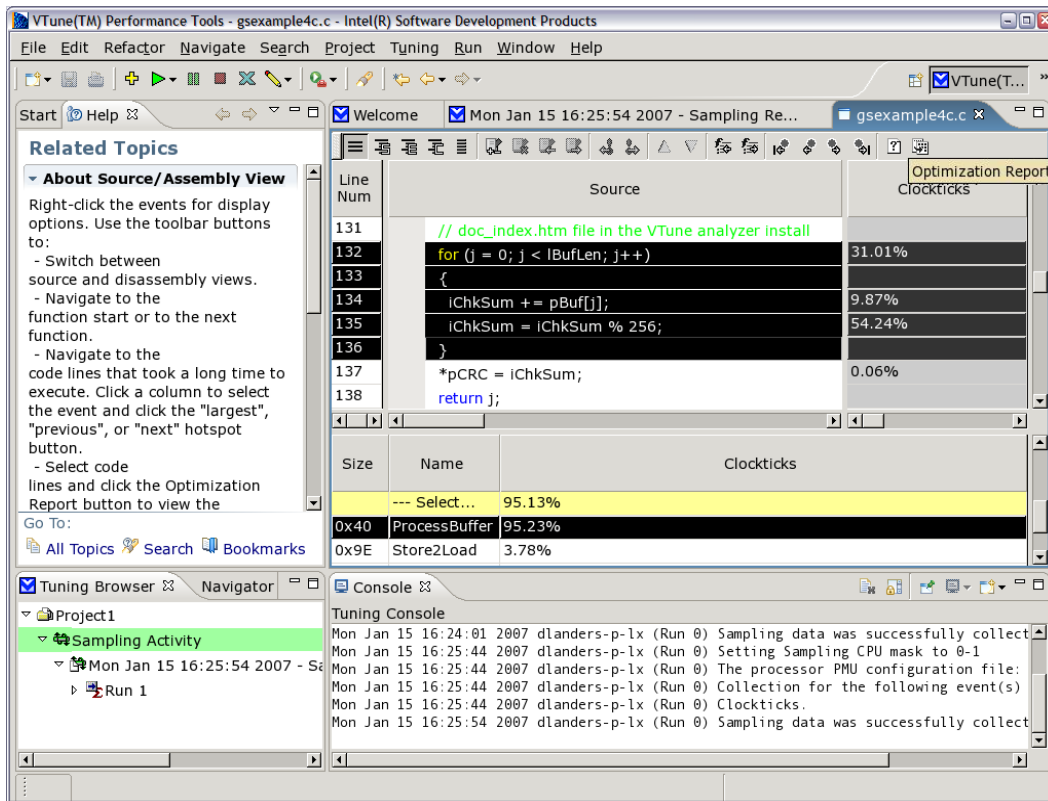


Figure 4: Selecting the hottest code lines and opening the Optimization Reports

A dialog opens and prompts for the optimization report file.

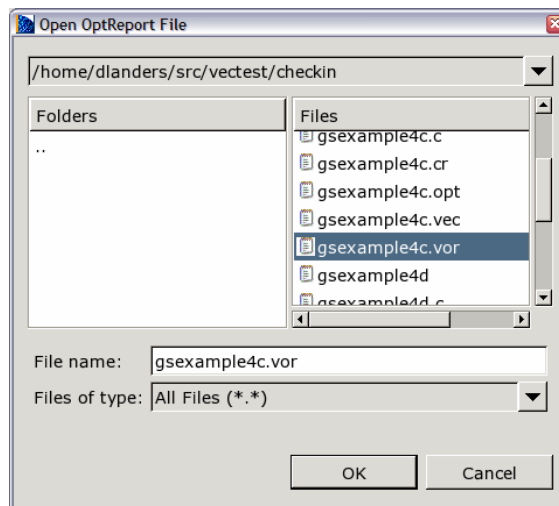
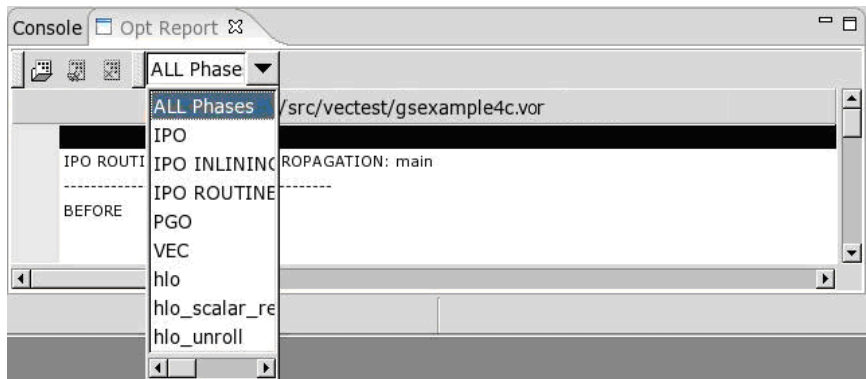


Figure 5: Selecting the Optimization Report file

Select the gsexample4c.vor file, the Optimization Report window opens. Select the VEC (vectorization) phase from the drop down list.



**Figure 6: Select the VEC phase of the report**

Vectorization automatically parallelizes code to maximize underlying processor capabilities. This advanced optimization analyzes loops and determines when it is safe and effective to execute several iterations of the loop in parallel by utilizing MMX™, SSE, SSE2, and SSE3 instructions. Vectorization is one of the premier ways that Intel compilers get the most performance out of the Intel® Core™2 Duo and Intel NetBurst® architecture-based processors ([more on vectorization](#)).

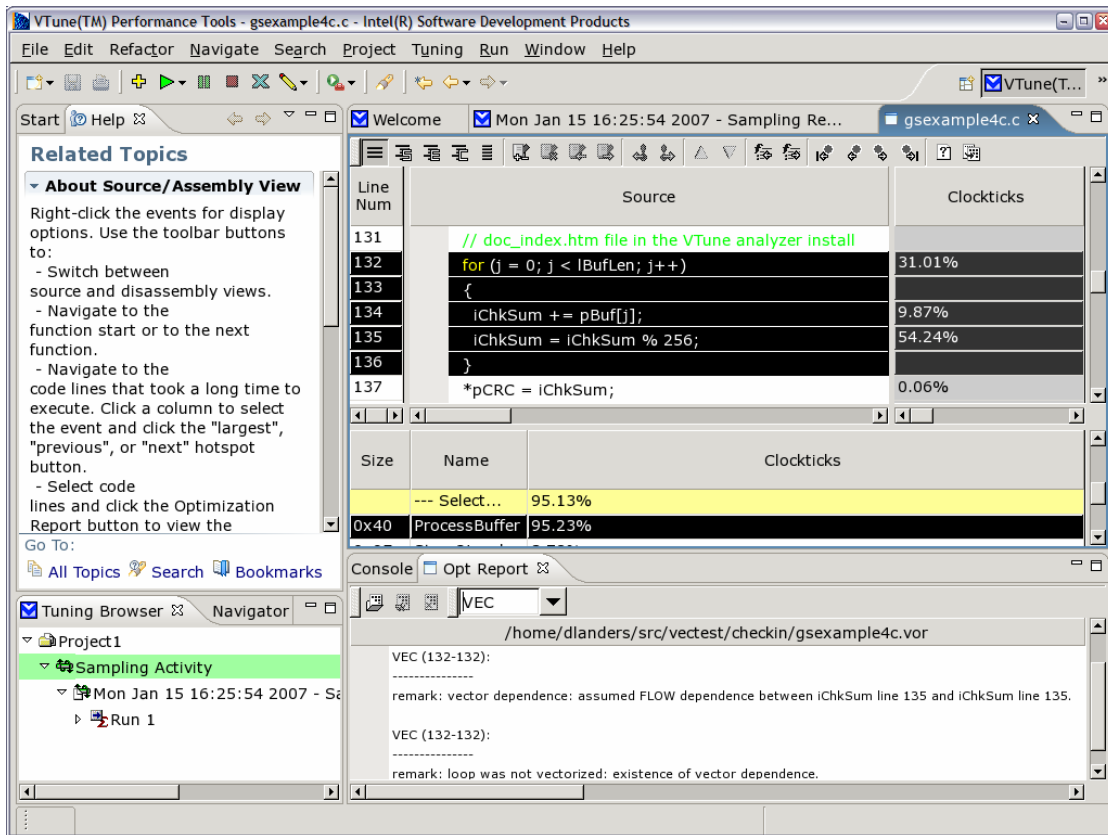


Figure 7: VEC phase report

In our example, the VEC report indicates that the loop was not vectorized because of a FLOW dependence of `iChkSum` ([more on loop dependencies](#)).

## 4 Optimizing the Code

At this point, examine the code and determine if you can change the code to enable the compiler to vectorize the loop. In this case, reviewing the algorithm reveals that the modulo part of the equation is required only as the final step of the function. Thus, you can change this code (in `gsexample4c.c`):

```

for (j = 0; j < lBufLen; j++)
{
    iChkSum += pBuf[j];
    iChkSum = iChkSum % 256;
}
*pCRC = iChkSum;

```



To this code (in `gsexample4d.c`):

```

for (j = 0; j < lBufLen; j++)
{
    iChkSum += pBuf[j];
}
*pCRC = iChkSum % 256;
    
```

This change enables the compiler to vectorize the loop. Verify this by building the `gsexample4d` application and generating the optimization and vectorization reports. The vectorization report shows:

```

gsexample4d.c(132) : (col. 2) remark: LOOP WAS VECTORIZED.
    
```

The resulting executable displays a dramatic increase in performance: 94,000 versus 37,600 loop iterations/sec on our system.

Finally, rerun the test under the VTune analyzer. You can see that the processing is more evenly distributed as the function `ProcessBuffer()` now only consumes ~48% of the total cycles vs. the ~80% in the 4c version.

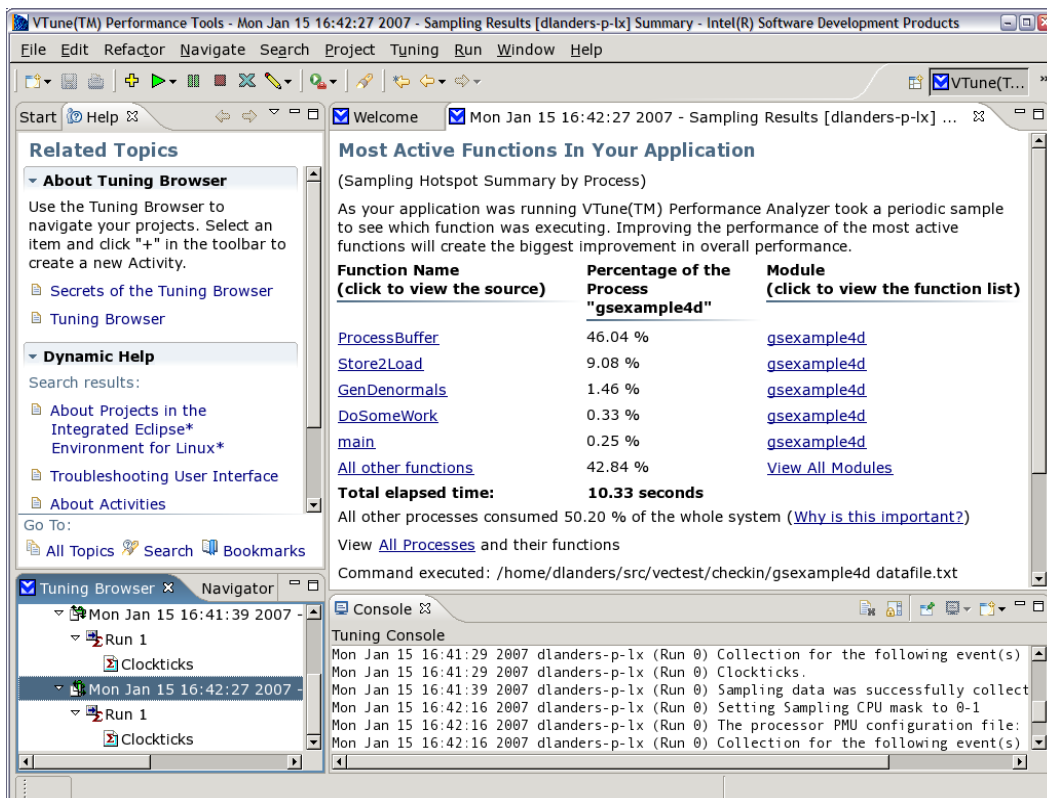


Figure 8: VTune analyzer results after vectorizing the application