**NVIDIA Devtech Compute**

**Performance Analysis Tools**

**For Internal Use Only**

**November 5, 2013**



**2701 San Tomas Expressway**

**Santa Clara, CA 95050**

Table of Contents

[1 Introduction 1](#_Toc371373548)

[1.1 Revisions 1](#_Toc371373549)

[1.2 Purpose 1](#_Toc371373550)

[1.3 Objectives 1](#_Toc371373551)

[1.4 File Names, Locations, and Descriptions 1](#_Toc371373552)

[1.5 Requirements 2](#_Toc371373553)

[2 Instructions for use 2](#_Toc371373554)

[2.1 Assumptions 2](#_Toc371373555)

[2.2 Gather the Profile Data 2](#_Toc371373556)

[2.2.1 Multi-node 3](#_Toc371373557)

[2.3 Import the Profile Data 3](#_Toc371373558)

[2.3.1 Importing into profile\_summary.xlsx 3](#_Toc371373559)

[2.3.2 Importing into profile\_analyzer.xlsx 3](#_Toc371373560)

[2.4 Submit your profile 4](#_Toc371373561)

[3 Confidentiality 4](#_Toc371373562)

# Introduction

## Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Modified By** | **Description** |
| 1.0 | 7/16/2013 | Cliff Woolley | Initial instructions |
| 1.1 | 8/1/2013 | Cliff Woolley | Explain data table recalculation delay |
| 1.2 | 10/23/2013 | Cliff Woolley | Pass count has been reduced to ~60 |

## Purpose

The Devtech Compute perf tools are a series of scripts and Excel spreadsheets that make use of the CUDA command line profiler to gather, summarize, analyze, and present data performance data about a CUDA or OpenCL application.

## Objectives

* To encapsulate and streamline the profile data analysis methods used by devtech for investigating the performance of kernels running on NVIDIA GPUs.
* To provide a level of detail greater than what can be provided in a public tool such as the NVIDIA Visual Profiler by way of experimental and/or internal-only performance data not available in the public tool.
* To provide a proving ground for new analysis metrics derived from the raw profile event data and/or new visualizations of this data that can incorporated into future versions of the public profiling tools.

## File Names, Locations, and Descriptions

The latest versions of the Devtech Compute perf tools are located in Perforce at //sw/devrel/tools/cudaperf . The files are as follows:

* runProfile.py :

This script along with its two child scripts collectData.py and summarizeProfile.py is what gathers the raw data and prepare it for import into the following Excel spreadsheets.

* profile\_summary.xlsx :

This spreadsheet uses a series of pivot tables to present an overview of the time spent in various routines in an overall application. It is useful for determining which kernels are the top kernels, the relative amounts of time spent in those kernels, and a few key summary metrics about the performance of each. If you already know which kernel(s) you want to look at, move directly on to the other spreadsheet:

* profile\_analyzer.xlsx :

This spreadsheet provides the bulk of the analysis logic for investigating kernel performance. It is generally used for analyzing the performance of the top kernels of an application in detail one at a time.

* selectnode.sh :

This is a helper script that can be used along with a multi-node job to profile just one of those nodes (which is all that is supported).

## Requirements

These tools require access to both a Python interpreter and Excel 2007 or later (2010 recommended).

The scripts use the (environment-variable-based) CUDA command line profiler, which requires re-execution of the target program many times to gather the various counters needed by the analysis spreadsheets. If your program runs for more than a few seconds, the ~60 separate program runs required by this method will take a prohibitive amount of time. If your app falls into this category, you should try changing the program temporarily (just for profiling purposes) to call cudaDeviceReset() and exit() immediately after the kernel you are interested in has executed at least once.

If the kernel of interest is very long running (several seconds), this is also problematic, as some profiler counters can overflow. To fix this, you can try artificially reducing the number of CTAs launched (as above, this would be a temporary change just for profiling purposes) to reduce kernel runtime.

[[ Aside: nvprof, which is the newer command-line-based profiling tool included with the CUDA Toolkit, supports a kernel replay mode in CUDA 5.5 and later, which has the potential to fix the multi-pass issues; right now, however, nvprof only gathers enough data to satisfy the Visual Profiler, not these spreadsheets. Still, that may be a viable alternative for many use cases. Use nvprof’s --analysis-metrics argument (in CUDA 5.5 or later) along with the -o [outputfile] argument to capture a trace that can be imported into NVVP. We have an open RFE ([1270656](http://nvbugs/1270656)) that would allow this mode of nvprof to be extended to cover our use case here; once that method is implemented, collectData.py will be updated to make use of it. ]]

# Instructions for use

## Assumptions

The scripts assume that repeated re-execution of the program is safe (i.e., it does not overwrite, modify, or delete its input from run to run, and any output file(s) it may create will be ignored and overwritten without issue by subsequent runs).

Also, the program execution *must* be deterministic; if its behavior is nondeterministic, then data collected in several independent trials will not properly correspond to each other, and the resulting profile will be broken/invalid.

## Gather the Profile Data

Gathering the profile data is easy; it is completely automated. Ensure that you have python available, then run your program (e.g., a.out) from the command prompt in a manner similar to the following:

**$** ./runProfile.py ./a.out arg1 arg2 ...

Here “arg1”, “arg2”, and so on are whatever command line arguments you would normally pass to ./a.out when executing it – they will be passed through to the application during profiling.

### Multi-node

If you are running on a multi-node/cluster system such as a Cray machine that uses a tool like aprun to issue work to multiple nodes, you will need to use a helper script like the provided **selectnode.sh** to choose just one of these nodes for gathering profile data. You use it like this:

**$** ./runProfile.py aprun ... ./selectnode.sh nid00040 ./a.out arg1 ...

## Import the Profile Data

The steps in section 2.1 will have created two CSV files called **profile\_summary\_in.csv** and **profile\_analyzer\_in.csv**. Use the following steps to import the data from these CSV files into the main spreadsheets:

### Importing into profile\_summary.xlsx

1. Open **profile\_summary.xlsx** and **profile\_summary\_in.csv** in Excel.
2. In **profile\_summary\_in.csv**, use Ctrl-A Select all of the data, then Ctrl-C to copy it to the clipboard.
3. In **profile\_summary.xlsx**:
   1. Switch to the **profile data** datasheet, select cell A1, and use Ctrl-V to paste in the profile data.
   2. Switch to the **Level Breakdown** or **Method Breakdown** datasheet. Press Alt-F5 to update the Pivot table with the recently imported data.

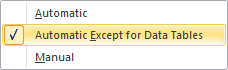
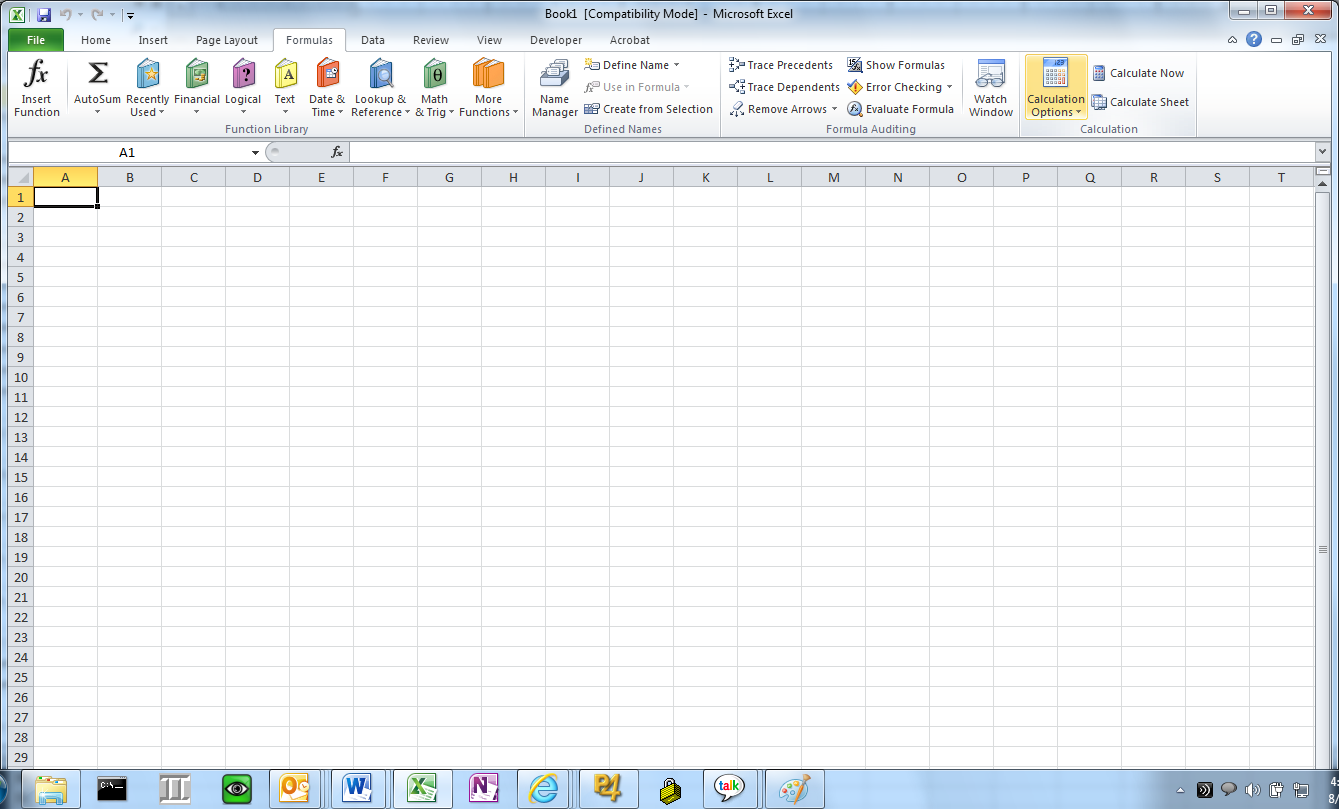
### Importing into profile\_analyzer.xlsx

1. Open **profile\_analyzer.xlsx** and **profile\_analyzer\_in.csv** in Excel.
2. In **profile\_analyzer\_in.csv**, use Ctrl-A Select all of the data, then Ctrl-C to copy it to the clipboard.
3. In **profile\_analyzer.xlsx**:
   1. Create a new datasheet.
   2. Name this new datasheet something meaningful w.r.t. the current application and/or version of that application, e.g. “S3D-v1”.
   3. Select cell A1 of this new datasheet and use Ctrl-V to paste in the profile data.
   4. Switch to the **Overview** datasheet.
   5. Type in the name of the datasheet you selected in step b where it says “enter name of data sheet to analyze”.
   6. Select the name of the kernel of interest from the pulldown list in the cell labeled “select kernel to analyze”. If you choose to type the kernel name, note that it must be written absolutely identically to the way it appears in the method column of profile\_analyzer\_in.xlsx.

Note: With this spreadsheet, you can only analyze one instance of a given kernel name. The Python scripts will have arranged it so that whichever instance of each kernel took the longest is the one that will be shown when you choose that name in the “select kernel to analyze” field.

Note: Kernels with very long names (such as can happen in C++-template-heavy code, like in Thrust) cause problems for this spreadsheet, so such names will have been truncated by the Python script at 100 characters. This may pose further problems due to creating false duplicate names; if so, rename each instance’s method name to be whatever you like in the input datasheet.

Note: The profile analyzer spreadsheet uses Excel’s “data table” feature to populate the graphs on the “Analysis – Occupancy” tab (which is basically the same as the CUDA Occupancy Calculator, except integrated with the rest of this system). That feature, while useful, can cause very long recalculation delays when you change unrelated fields. To fix this, select to the Formulas ribbon, click on Calculation Options, and select “Automatic Except for Data Tables”:



However, doing this means that you will need to explicitly tell Excel when it should update the data tables. Whenever you change something related to occupancy and want to see the occupancy graphs get updated, press Shift-F9 from the Analysis – Occupancy tab, and that will trigger the recalculation.

## Submit your profile

As a totally optional final step, please consider contributing the profile of your application (even intermediate profiles of partially optimized or unoptimized codes are useful) into the spreadsheet so that we can build up our library of examples. You can either add the profiles as new worksheets (suitably named) and submit the spreadsheet directly to Perforce yourself or, if you prefer, email your .csv files (along with at least the name of the app and a few words about whatever insights you had about it) to Cliff Woolley ([jwoolley@nvidia.com](mailto:jwoolley@nvidia.com)) for inclusion.

# Confidentiality

These performance analysis tools and spreadsheets are for ***NVIDIA INTERNAL USE ONLY*** and **may not be distributed outside NVIDIA, even under NDA**.