Performance Benchmarking the Insight Toolkit

Release 1.0.0

Matthew McCormick[1], Hyun Jae Kang[1], and Sebastien Barre[1]

July 1, 2016

[1]Kitware, Inc, Carrboro, NC

Abstract

This document describes a module for the Insight Toolkit (ITK) www.itk.org to assist in performance benchmarking and a suite of benchmarks for the toolkit. These resources provide metrics to quantify software computational performance. This is a prerequisite to improve performance either through algorithmic advancements or better utilization of hardware resources.

Computational performance is quantified by reducing factors that confound timing measurements and by estimating measurement variance. New classes are presented that increase operating system process priority to minimize the impact of other processes running on the system. System hardware characteristics are extracted and displayed. The influence of hard disk input/output on runtime measurements is removed in the suite of benchmarks. Additionally, the number of threads used by each benchmark can be specified. These benchmarks consist of common analysis pipelines and run on 3D magnetic resonance brain image data so realistic performance is quantified. Benchmarks can be executed with multiple iterations, and timing statistics are recorded in tab separated value (.tsv) files, which are easily stored or processed for further analysis and visualization.

This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the algorithm described in this paper. This adheres to the fundamental principle that scientific publications must facilitate reproducibility of the reported results.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3557]

Contents

1 Introduction **2**

2 PerformanceBenchmarking Module **2**

3 Benchmark Suite **4**

3.1 The Benchmarks 4

3.2 How to Build and Run 6

1 Introduction

To make quantifiable improvements in computational software performance, the performance must be measurable. This article describes performance measurement tools in the Insight Toolkit (ITK,www.itk.org), and how they are applied to benchmark scientific image analysis code. In addition, the article also describes a suite of example benchmarks that quantify ITK image processing pipeline performance. Informative performance metrics require consideration of the behavior to be optimized and factors that influence measurement of the desired behavior

The behavior this article focuses on is the time required to execute software image analysis pipelines on realistic image datasets. Full pipelines are benchmarked since this is what impacts the experience of software users. Hard disk input/output (I/O) influence is removed from timings because it may not be present in applications in the same form as the benchmarks. The data processed is based on a 3D magnetic resonance BrainWeb[2, 1] image, which has realistic size and content. Benchmarks inputs and operations are also designed for a target runtime of approximately 1-2 seconds for a single iteration and approximately 5-10 seconds for multiple iterations on current systems. These durations are intended to be long enough for memory access and processing times to reach a steady state yet short enough for the benchmark suite to be executed in a reasonable amount of time.

The benchmarks attempt to minimize the influence of the operating system and hardware and other processing running on the system. The type of operating system and hardware used is recorded and the number of hardware threads used is governable in each benchmark so performance can be normalized and optimized for processor and memory type and thread-based parallelism. Operating system process priority is increased on the benchmark process to reduce the impact of other processes running on the system. A high precision real-time clock is used to measure timings. Timing variations are recorded in statistics generated by running the benchmark pipelines multiple times.

The code for the ITK module, example benchmarks, and results presented in this article can be found at `https://github.com/InsightSoftwareConsortium/ITKPerformanceBenchmarking`. The code must be built against ITK 4.11.0 or newer, which includes improvements to the resource recording classes.

The sections that follow describe in further detail first the classes in the `PerformanceBenchmarking` ITK module and how they are used, then the example benchmark suite.

2 PerformanceBenchmarking Module

The `PerformanceBenchmarking` module currently contains classes that extend the `itk::RealTimeClock`, `itk::RealTimeProbe`, and `itk::RealTimeProbesCollector` classes by also increasing the current process's operating system priority. The `itk::HighPriorityRealTimeClock` class uses operating system specific calls to increase the current process's priority in its constructor and return it to its previous value in its destructor. The `itk::HighPriorityRealTimeProbe` and `itk::HighPriorityRealTimeProbesCollector` use a `itk::HighPriorityRealTimeClock`.

A higher process priority increases the amount of computing resources allotted to the benchmark process relative to the other processes running on the system.

Figure 1 illustrates the difference in benchmarks timings when an ITK build is running in the background on the system and when only standard background processes are running. When only standard background processes are running, the timing results are similar regardless of the process priority. However, an elevated process priority reduces the increased mean time and variance that occurs with a high system background

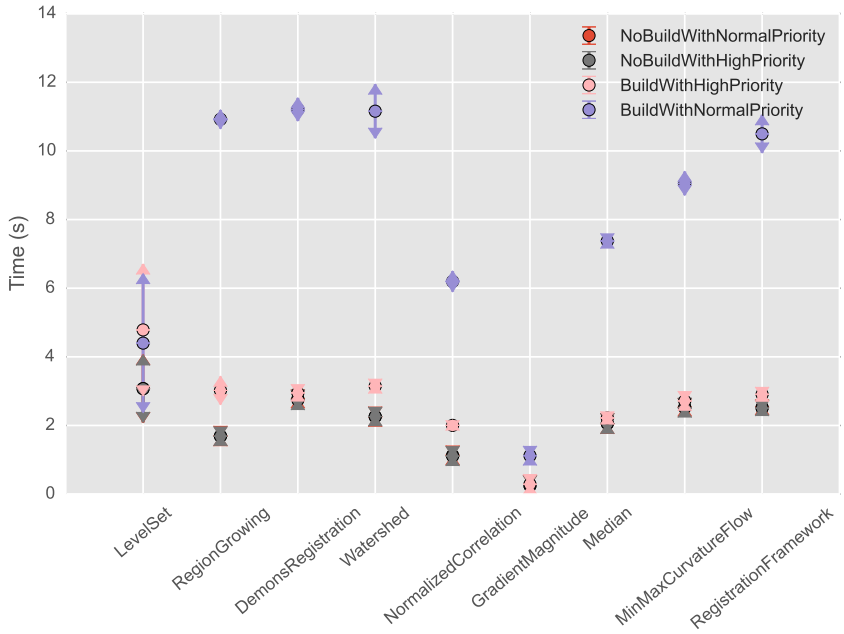

Figure 1: The impact of an increased process priority when a parallel build of ITK is running in the background and when only a few system processes are running. Times are given with mean +/- two standard errors. In general, benchmark run times and their variability is more closer to low load system results when the process priority is increased.

load. The parallel builds were executed with the `-j20` parallelism flag on a system whose characteristics are given below.

```
System:              meyer
Processor:           Unknown P6 family
    Cache:           6144
    Clock:           3250.41
    Physical CPUs:   1
    Logical CPUs:    8
    Virtual Memory:  Total: 19322          Available: 19321
    Physical Memory: Total: 16003          Available: 7149
OSName:              Linux
    Release:         3.16.0-4-amd64
    Version:         #1 SMP Debian 3.16.7-ckt25-2 (2016-04-08)
    Platform:        x86_64
    Operating System is 64 bit
ITK Version: 4.11.0
```

Example usage of the high priority timing classes is shown in the code below, taken from the Median example benchmark. Note that this may effect other high priority applications running on the system like graphical user interfaces, network connections, etc. On Unix-like system, the process should be run as the `root` user so it has adequate permission to elevate the priority; running as a normal user will not result in a higher than normal process priority.

```
#include "itkHighPriorityRealTimeProbesCollector.h"

[...]

ImageType::Pointer inputImage = reader->GetOutput();
inputImage->DisconnectPipeline();

itk::HighPriorityRealTimeProbesCollector collector;
for( int ii = 0; ii < iterations; ++ii )
  {
  inputImage->Modified();
  collector.Start("Median");
  filter->UpdateLargestPossibleRegion();
  collector.Stop("Median");
  }
bool printSystemInfo = true;
bool printReportHead = true;
bool useTabs = false;
collector.Report( std::cout, printSystemInfo, printReportHead, useTabs );

std::ofstream timingsFile( timingsFileName, std::ios::out );
printSystemInfo = false;
useTabs = true;
collector.ExpandedReport( timingsFile, printSystemInfo, printReportHead, useTabs );
```

3 Benchmark Suite

3.1 The Benchmarks

The `examples` directory of the `ITKPerformanceBenchmarking` repository contains a suite of ITK image processing pipelines. The purpose of these benchmarks is to be a timing metric for ITK processing pipelines. They are intended to help monitor computational performance as changes are made to the toolkit and to measure the effect of internal changes to pipeline code. To apply these benchmarks for comparisons of different algorithmic approaches with the same analytic objective instead, greater care should be given to ensure parameters of the pipelines are optimal. Additionally, attention is required to establish that the results from competing algorithms are similar.

The benchmarks currently cover a variety of common processing pipelines that require different parts of the toolkit. Additional benchmarks could be added to benchmark other processing objectives and implementations in the toolkit. Table 1 describes the current benchmarks, their processing objective, and the features of the toolkit they exercise.

Image data used as inputs to the benchmarks were derived from the image shown in Figure 2.

When executed, each benchmark outputs system information and summary statistical information. For example,

```
System:             meyer
Processor:          Unknown P6 family
    Cache:          6144
    Clock:          3212.72
    Physical CPUs:  1
    Logical CPUs:   8
    Virtual Memory:  Total: 19322          Available: 19322
    Physical Memory: Total: 16003          Available: 6674
```

Type	Benchmark Name	Objective	Toolkit Features Exercised
Filtering	GradientMagnitude	Emphasize image edges	itk::GradientMagnitudeRecursiveGaussianImageFilter, itk::NeighborhoodInnerProduct, multi-threaded itk::ImageToImageFilter
	Median	Image denoising through median filtering	itk::MedianImageFilter, itk::BoxImageFilter, multi-threaded itk::ImageToImageFilter
	MinMaxCurvatureFlow	Curvature driven image denoising	itk::MinMaxFlowCurvatureImageFilter, itk::DenseFiniteDifferenceImageFilter
Registration	DemonsRegistration	Demons image registration	itk::PDEDeformableRegistrationFilter, itk::DemonsRegistrationFilter
	NormalizedCorrelation	Normalized cross correlation image registration	itk::FFTNormalizedCorrelationImageFilter, itk::FFTPadImageFilter, itk::ForwardFFTImageFilter, itk::InverseFFTImageFilter, itk::MinimumMaximumImageCalculator
	RegistrationFramework	Optimized cost function image registration	ITKv4 image registration framework, itk::MeanSquaresImageToImageMetricv4
Segmentation	LevelSet	Level set segmentation with a propagation and curvature term	ITKv3 level set framework, itk::ShapeDetectionLevelSetImageFilter, itk::CurvatureAnisotropicDiffusionImageFilter, itk::GradientMagnitudeRecursiveGaussianImageFilter, itk::FastMarchingImageFilter, itk::BinaryThresholdImageFilter
	RegionGrowing	Region growing segmentation	itk::ConfidenceConnectedImageFilter, itk::CurvatureFlowImageFilter, itk::BinaryFillholeImageFilter
	Watershed	Watershed segmentation	itk::WatershedImageFilter, itk::CurvatureFlowImageFilter, itk::GradientMagnitudeRecursiveGaussianImageFilter, itk::RelabelComponentImageFilter

Table 1: Benchmarks in the example benchmark suite.

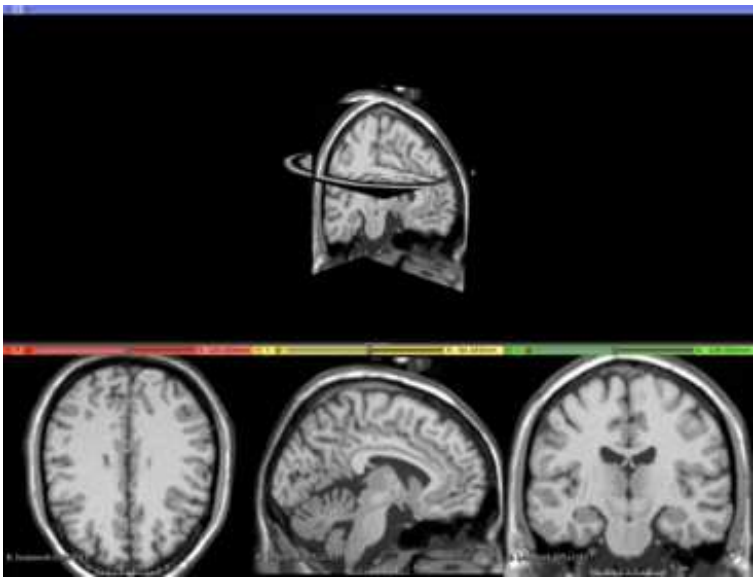

Figure 2: Input BrainWeb [2, 1] image data used in the benchmarks.

```
OSName:             Linux
    Release:        3.16.0-4-amd64
    Version:        #1 SMP Debian 3.16.7-ckt25-2 (2016-04-08)
    Platform:       x86_64
    Operating System is 64 bit
ITK Version: 4.11.0
Name Of Probe (Time)          Iterations     Total (s)      Min (s)        Mean (s)       Max (s)        StdDev (s)
Median                        3              6.19352        2.05278        2.06451        2.08378        0.0168256
```

Additionally, extended summary statistics are recorded in a tab separated value file. For example,

```
Name Of Probe (Time) Iterations Total (s) Min (s) Mean-Min (diff) Mean/Min (%) Mean (s) Max-Mean (diff) Max/Mean (%) Max (s) Total Diff (s) StdDev (s)
StdErr (s)
Median 3 6.19352 2.05278 0.011723 100.571 2.06451 0.019279 100.934 2.08378 0.031002 0.0168256 0.00971427
```

3.2 How to Build and Run

The benchmark suite requires ITK 4.11.0 or newer and the PerformanceBenchmarking module. The repository `examples/CMakeLists.txt` file will configure and build both these dependencies by default. To build the benchmarks:

```
$ git clone https://github.com/InsightSoftwareConsortium/ITKPerformanceBenchmarking
$ mkdir Benchmarks-build
$ cd Benchmarks-build
$ cmake ../ITKPerformanceBenchmarking/examples/
$ make
```

The benchmarks can be executed by running the `ctest` executable. When the benchmark project is built, it also downloads required input data.

```
$ cd ITKBenchmarks-build/
$ ctest
Test project /home/matt/Benchmarks-build/ITKBenchmarks-build
      Start  1: MedianBenchmark
 1/10 Test  #1: MedianBenchmark .....................   Passed    6.24 sec
      Start  2: GradientMagnitudeBenchmark
 2/10 Test  #2: GradientMagnitudeBenchmark ..........   Passed    4.10 sec
      Start  3: GradientMagnitudeBenchmark1Thread
 3/10 Test  #3: GradientMagnitudeBenchmark1Thread ...   Passed    4.22 sec
      Start  4: MinMaxCurvatureFlowBenchmark
 4/10 Test  #4: MinMaxCurvatureFlowBenchmark ........   Passed    7.63 sec
      Start  5: RegistrationFrameworkBenchmark
 5/10 Test  #5: RegistrationFrameworkBenchmark ......   Passed    7.67 sec
      Start  6: DemonsRegistrationBenchmark
 6/10 Test  #6: DemonsRegistrationBenchmark .........   Passed    8.50 sec
      Start  7: NormalizedCorrelationBenchmark
 7/10 Test  #7: NormalizedCorrelationBenchmark ......   Passed    3.37 sec
      Start  8: RegionGrowingBenchmark
 8/10 Test  #8: RegionGrowingBenchmark ..............   Passed    5.20 sec
      Start  9: WatershedBenchmark
 9/10 Test  #9: WatershedBenchmark ..................   Passed    6.88 sec
      Start 10: LevelSetBenchmark
10/10 Test #10: LevelSetBenchmark ...................   Passed    9.26 sec

100% tests passed, 0 tests failed out of 10

Label Time Summary:
Filtering       =  22.06 sec (4 tests)
Registration    =  19.55 sec (3 tests)
Segmentation    =  21.34 sec (3 tests)

Total Test time (real) =  63.08 sec
```

To see how a benchmark is called, run `ctest` with the `-V` flag. The first argument is always the path to the output tab-separated-value file, the second argument is the number of iterations to run the benchmark pipeline, and the third argument is the number of threads to use; a value less than one will use as many threads as there are logical cores available on the system.

References

[1] C.A. Cocosco, V. Kollokian, R.K.-S. Kwan, and A.C. Evans. Brainweb: Online interface to a 3d mri simulated brain database. In *Proceedings of 3-rd International Conference on Functional Mapping of the Human Brain*, volume 5, page S425, 1997. 1, 2

[2] McGill University McConnell Brain Imaging Centre, Montreal Neurological Institute. *BrainWeb: Simulated Brain Database*. http://brainweb.bic.mni.mcgill.ca/brainweb/, 2010. 1, 2