Itrace, an Iterator Tracing Tool

Michael LaSpina David R. Musser

Draft, April 20, 2002

1. Introduction

Itrace is a tool for analyzing the memory access patterns of generic algorithms in C++. It provides an iterator adaptor that maintains a log of time vs. position data, which can then be output to a file and fed directly to a program such as Gnuplot for viewing. Itrace also provides facilities for comparing different algorithms (e.g. overlayed graphs), as well as different data structures and their impact on the performance of algorithms.

2. Usage

Itrace is controlled through the use of several command line options. Below is a brief overview of each of the available options.

Option	**Meaning**
h	Help. Provides a lising of all the support algorithms and containers, as well as the options listed here.
c	Adds a container to the set of containers to be tested. The default is `vector`
gc	Adds a graph container to the set of containers to be tested.
a	Adds an algorithm to the set of algorithms to be tested. The default is `sort`
ga	Adds a graph algorithm to the set of algorithms to be tested.
n	Sets the size of the data set. The default is 1000.
vertices	Sets the number of vertices in the graph. The default is 1000.
edge	Sets the number of edges in the graph. The default is 1000.
v	Specifies an additional argument to be used by an algorithm. This is algorithm specific.
e	External data. Itrace will try to read data values from stdin, if there aren't enough, random values will be used.
r	Reuses the same data set for all tests.
f	Sets the data filter. The default is to filter out time gaps and to adjust the data so the initial time is zero.
o	Changes the file name for the gnuplot commands that are generated. The default is results.plot.
s	Changes the plot style. This must be a valid Gnuplot style. The default is "dots."

2.1. Algorithms and Containers

For each algorithm you would like Itrace to analyze add one `-a <algorithm>` argument, where `<algorithm>` is the name of the algorithm to test. For example:

```
$ ./itrace -a sort -a stable_sort
```

will run both `sort` and `stable_sort` on vectors of 1000 elements. For each test that Itrace runs, it produces a text file containing the time versus position data. Additionally, Itrace will produce a file that contains a series of Gnuplot commands. This file can be passed to Gnuplot to produce an overlayed graph of all the data sets Itrace produced.

Containers work in much the same way. So,

```
$ ./itrace -c vector -c deque
```

will run `sort` using a `vector` and `deque` with 1000 elements each. It will produce one data file for each container, and also a file containing Gnuplot commands.

Finally, multiple containers and algorithms can be combined.

```
$ ./itrace -a sort -a stable_sort -c vector -c deque
```

Will test `sort` and `stable_sort` with both `vector` and `deque` respectively. Producing one data file for each combination and also a set of Gnuplot commands.

2.2. Working with Graphs

Itrace provides support for working with the generic graph algorithms provided by the Boost Graph Library (BGL). Testing graph algorithms with Itrace is very similar to working with sequence algorithms. However, the names of the program options are slightly different. For example, to test Dijkstra's Shortest Paths algorithm on an adjacency list represtation of a graph with 1000 vertices and 1000 edges:

```
$ ./itrace -gc adj_list -ga dijkstra -vertices 1000 -edges 1000
```

The graph will default to 1000 vertices or 1000 edges if either parameter is left out, but there is neither a default graph container nor a default graph algorithm.

2.3. Data Options

Itrace provides several options for control over the data sets used during testing. The simplest is `-n`, which controls the number of elements in each container tested. When using multiple algorithms or containers often it is desirable to use the same data set in each test. Itrace provides this with the `-r` option. For example:

```
$ ./itrace -r -a sort -a stable_sort
```

This will test `sort` and `stable_sort` using the same data set. This option is particularly useful for comparing worst case behavior. Another option helpful for testing worst case behavior is `-e`, which allows Itrace to retrieve values from standard input. Typically, these values are redirected from a file or piped in from another program. For example:

```
$ ./itrace -e -a sort -a stable_sort < values.txt
```

The above snippet will test `sort` and `stable_sort` using the first 2000 values in the file *values.txt* (1000 for each test). If the input source does not contain enough values Itrace will display a warning that no more values were available from standard input and will retrieve any additional values from its random number generator.

Often, the data generated by Itrace will contain time gaps. Some of this is due the additional overhead logging introduces, but some of it is due to external factors. For example, events such as cache misses and virtual memory paging may be caused by other processes. Depending on the load on the system, this can introduce large gaps in the graphs produced by Itrace. To alleviate this, Itrace can filter data sets to remove these gaps before ouputting the results. This is performed using the `-f <filter>` switch, where `<filter>` is the name of the filter to be used. Below is a table of all the filtering modes support:

Filter	**Action**
simple	Removes time gaps in the data set according to a threshold. The threshold is a compile time constant and defaults to 5.
normalized	Adjusts the data set so that its initial time is zero.
raw	No filtering is performed.
default	Both simple and normalized modes are used.

For simple mode filtering the threshold can be controlled by defining the constant `ITRACE_THRESHOLD` to the desired value. It's recommended this value be defined in the Itrace makefile.

Some algorithms have need for additional parameters. For example, `partial_sort` takes three parameters. Two of the parameters denote the sequence of valued to sort, and the third denotes how many of the values should be sorted. Thus, `partial_sort` can be used to sort the first half of a sequence or the first third, etc, depending on the value of third parameter. Itrace provides the **-v** option for supplying an additional argument to an algorithm. In the case of `partial_sort`, we can supply an additional argument to indicate how many of the value we wish to sort. This will generate an iterator trace of sorting the first 500 values in a 1000 element sequence.

```
$ ./itrace -a partial_sort -v 500
```

Most algorithms do not require an additional argument, and those that do have appropriate defaults. Unneeded arguments are ignored by algorithms that do not use them. Unlike all of the other command line parameters accepted by Itrace, additional arguments are sensitive to their order. Since they need to be associated with a specific algorithm, any **-v** option must immediately follow the algorithm it applies to. For example, the following uses are incorrect:

```
$ ./itrace -v 500 -a partial_sort
$ ./itrace -a partial_sort -c deque -v 500
```

Instead they should appear as:

```
$ ./itrace -a partial_sort -v 500
$ ./itrace -a partial_sort -v 500 -c deque
```

The following table presents the algorithms that make use of additional arguments and their meaning.

Algorithm	**Type**	**Meaning**
`partial_sort`	Integer	Indicates the number of values to sort.
`nth_element`	Integer	Which value in the sequence to partition around.
`rotate`	Integer	Indicates how many positions to rotate each value.