Title: Automating the efficiency comparison of programs

Speaker: Jyrki Katajainen

Department of Computing
University of Copenhagen

Current status: Basically, I have not done anything else but read Python manuals.

Today: I will explain my ideas. Can you see any new features that should be in the system?
#include <iterator> // defines std::iterator_traits

// SGI STL find here renamed as STL_find

template <typename position, typename element>
inline position STL_find (  
    position p,
    position one_past_the_end,
    const element& value,
    input_iterator_tag
) {
    while (p != one_past_the_end && *p != value) {
        ++p;
    }
    return p;
}

template <typename position, typename element>
inline position STL_find (  
    position first,
    position one_past_the_end,
    const element& value
) {
    return STL_find(first, one_past_the_end, value,  
                     std::iterator_traits<position>::iterator_category());
}
template <typename position, typename element>
position STL_find (  
    position p,
    position one_past_the_end,
    const element& value,
    random_access_iterator_tag
) {

    std::iterator_traits<position>::difference_type
    trip_count = (one_past_the_end - p) >> 2;

    for ( ; trip_count > 0 ; --trip_count) {
        if (*p == value) return p;
        ++p;
        if (*p == value) return p;
        ++p;
        if (*p == value) return p;
        ++p;
        if (*p == value) return p;
        ++p;
    }

    switch(one_past_the_end - p) {  
    case 3:  
        if (*p == value) return p;
        ++p;
    case 2:  
        if (*p == value) return p;
        ++p;
    case 1:  
        if (*p == value) return p;
        ++p;
    case 0:  
        default:  
            return one_past_the_end;
    }
    
}
template<typename position, typename element>
position STL_find (  
    position first,  
    position one_past_the_end,  
    const element& value,  
    bidirectional_iterator_tag // plus something extra  
) {

    if (first == one_past_the_end) {  
        return one_past_the_end;  
    }

    position last = one_past_the_end;  
    --last;  
    element temp = *last;  
    *last = value;

    position p = first;  
    while (*p != value) {  
        ++p;  
    }

    *last = temp;  
    if (p == last && temp != value) {  
        return one_past_the_end;  
    } else {  
        return p;  
    }
}
Unsuccessful search from an unsorted array of n unsigned integers

Execution time per element [in nanoseconds]

SGI STL find
SGI STL find with specialization
Sentinel technique

Integers; Many Arrays; Pentium
Unsuccessful search from an unsorted array of $n$ unsigned integers.
Unsuccessful search from an unsorted array of n unsigned integers

Execution time per element [in nanoseconds]

n

SGI STL find
SGI STL find with specialization
Sentinel technique
Unsuccessful search from an unsorted array of \( n \) unsigned integers

- SGI STL find
- SGI STL find with specialization
- Sentinel technique

**Execution time per element [in nanoseconds]**

- \( 1 \) nanosecond
- \( 10 \) nanoseconds
- \( 100 \) nanoseconds
- \( 1000 \) nanoseconds
- \( 10000 \) nanoseconds
- \( 100000 \) nanoseconds
- \( 1000000 \) nanoseconds

**n**

- \( 10 \)
- \( 100 \)
- \( 1000 \)
- \( 10000 \)
- \( 100000 \)
- \( 1000000 \)
Unsuccessful search from an unsorted array of \( n \) unsigned integers
Directory Structure

makefile

Driver:
do_nothing.cc    uint-driver-many.cc
do_nothing.o      uint-driver-one.cc
string-driver-many.cc

Figure:
freja-uint-many.ps nanna-string-many.ps
guldfaxe-uint-many.ps nanna-uint-many.ps
loke-uint-many.ps  nanna-uint-one.ps

Gnuplot:
plot.gp

SGI-in:
drive    find.cc    find.o    plot.data

SGI-ran:
drive    find.cc    find.o    plot.data

Script:
find.sh

Sentinel:
drive    find.cc    find.o    plot.data

core:
Goal

- write the programs to be compared
- fill a form or two
- generate the plots automatically
Example Form

Start execution: 23.10.2000 02:00
Notification: e-mail

Compiler: g++
Compiler options: -O4 -Wall

Computers: nanna freja loke

Sequence type: (unsigned int)[]
Sequence form: increasing

Clock precision: 1 s
Repetitions: distinct sequences

Include files: find1.cc find2.cc find3.cc
Routine names: find find find
Routine texts: "SGI STL find" "SGI STL find with \specialization" "Sentinel technique"
\# parameters: 3
1st: first; step: 2^{i}, i \in [4, 5 .. 22]
2nd: first + 2^{i}; step: 2^{i}, i \in [4, 5 .. 22]
3rd: const 2^{22}

Plot title: "Unsuccessful search: n unsigned integers"
x-label: "n"
y-label: "Execution time per element [in nanoseconds]"
x-axis: 2^{i}, i \in [4, 5 .. 22]
y-axis: time / x
x-scale: logarithmic
x-range: [10 .. 2^{22}]
x-tics: ["10" 10, "100" 100, "1 000" 1000, "10 000" 10000, "100 000" 100000, "1 000 000" 1000000]
Your Written Projects

- write alternative implementations of your module
- use my tool to generate plots
- use Doxygen to generate documentation of your programs
- use \LaTeX to write a short report about your work; we provide you a template file.
BEGINNING OF MY MAKEFILE

MAXSIZE = 4194304  # The maximum length of the test array

CXX = g++  # Compiler used
CXXFLAGS = -O4 -Wall -DMAXSIZE=$(MAXSIZE)

PROGRAMS = SGI-in/drive SGI-ran/drive Sentinel/drive
DRIVER = Driver/uint-driver-one.cc

all: $(PROGRAMS)

plot: plot.ps

plot.ps: Gnuplot/plot.gp SGI-in/plot.data\ 
   SGI-ran/plot.data Sentinel/plot.data
gnuplot Gnuplot/plot.gp

Driver/do_nothing.o: Driver/do_nothing.cc
$(CXX) Driver/do_nothing.cc -c -o Driver/do_nothing.o

# SGI STL find with input iterators

SGI-in/find.o: SGI-in/find.cc
$(CXX) $(CXXFLAGS) -c -o SGI-in/find.o SGI-in/find.cc

SGI-in/drive: $(DRIVER) Driver/do_nothing.o SGI-in/find.o
$(CXX) $(CXXFLAGS) -ISGI-in $(DRIVER) -o SGI-in/drive

SGI-in/plot.data: Script/find.sh SGI-in/drive
Script/find.sh SGI-in > SGI-in/plot.data #2> /dev/null
Kernel of a Driver

// Dual loop

generate(first, one_past_the_end);

clock_t start = clock();
for (volatile unsigned int k = 0; k < repetitions; k++) {
    volatile position beg = first + k * n;
    volatile position end = first + (k+1) * n;
    element x = element(MAXSIZE);
    volatile position answer = do_nothing(beg, end, x);
    assert(answer == end);
}
clock_t extra = clock() - start;

start = clock();
for (volatile unsigned int k = 0; k < repetitions; k++) {
    volatile position beg = first + k * n;
    volatile position end = first + (k+1) * n;
    element x = element(MAXSIZE);
    volatile position answer = search(beg, end, x);
    assert(answer == end);
}
clock_t ticks = clock() - start - extra;

double ns_per_n = 1.0e9 / (double(CLOCKS_PER_SEC) * \
    double(n));
printf("%ld \t %f\n", n, ns_per_n * double(ticks) / \
    double(repetitions));
Shell Script

#!/usr/local/bin/tcsh -f

# Modified from the script written by:
# Jesper Bojesen
# e-mail: Jesper.Bojesen@uni-c.dk

# Version: 1.0
# May 2000

#################################

c

nice

#.drive <n> <type> <seed>

$1/drive 10 1
$1/drive 16 1
$1/drive 32 1
$1/drive 64 1
$1/drive 128 1
$1/drive 256 1
$1/drive 512 1
$1/drive 1024 1
$1/drive 2048 1
$1/drive 4096 1
$1/drive 8192 1
$1/drive 16384 1
$1/drive 32768 1
$1/drive 65536 1
$1/drive 131072 1
$1/drive 262144 1
$1/drive 524288 1
$1/drive 1048576 1
$1/drive 2097152 1
$1/drive 4194304 1

© Performance Engineering Laboratory
# Modified from the script written by:
# Jesper Bojesen
# e-mail: Jesper.Bojesen@uni-c.dk

# Version: 1.1
# October 2000

#########################################

set xrange [10:4194304]
set logscale x

set xtics ( "10" 10, \
    "100" 100, \
    "1 000" 1000, \
    "10 000" 10000, \
    "100 000" 100000, \
    "1 000 000" 1000000 \
)

set output "Figure/plot.ps"
set terminal postscript landscape monochrome dashed
set title 'Unsuccessful search from an unsorted array of n unsigned integers'
set xlabel 'n'
set ylabel 'Execution time per element [in nanoseconds]'
plot "SGI-in/plot.data" title "SGI STL find" with \
    linespoints 1 1,\n"SGI-ran/plot.data" title "SGI STL find with specialization" with linespoints 3 3,\n"Sentinel/plot.data" title "Sentinel technique" \n    with linespoints 4 4,\n0 notitle