WaveScript Benchmarks Performance Report

August 20, 2008

Machine information:
Linux chastity 2.6.22-14-generic #1 SMP Tue Feb 12 07:42:25 UTC 2008 i686 GNU/Linux

WaveScript SVN:
Revision: 3518

WaveScope Engine SVN:
(omitted for now)

1 Microbenchmarks

This section reports various microbenchmarks that stress the implementation of particular language constructs or data types.
Per-stream-element overheads

One thing that you can see, is that currently (2007.10) the C++/XStream engine has a high per-tuple (that is, per-element) on the communication channels relative to the ML backend. The just_timer test stresses this, doing nothing but passing a large number of unit tuples.

Focusing on scheduling overheads a bit more, we turn to the following data passing microbenchmarks. These do nothing but generate a stream of numbers, and then add up windows of those numbers. We vary the window size in the following graphs. The numbers are passed either one at a time (“raw”), or in bulk using arrays or lists.

Notes:
• FFT results for Scheme above depend on whether or not it is configured to use FFTW, or a native Scheme fourier transform.

2 Language Shootout Benchmarks

This is where I will accumulate some of the small benchmarks from the language shootout. Here are some per-benchmark comments:

• fannkuch - “pancake flipping”. This is a translation of the gcc version of the benchmark. Tests indexed access to a small array.

3 Application Benchmarks

This section includes performance results on larger programs, namely, our current applications. Presently (2007.10) the largest of these by far is the marmot application.

3.1 Marmot Application

We start off by looking at the original, hand-optimized marmot application that we deployed.
4 Data Representation Profiling

This is stale data for now... having sneaky problems with the datarep Makefile that are hosing regression tests. [2007.11.07]

This section includes an analysis of the efficiency of different data representations under different backends. This should theoretically be run on different hardware platforms as well (such as the ARM-based ensboxes).

4.1 Arrays of Arrays

Arrays of arrays are notable because they cannot generally be flattened (the inner arrays will always be pointers). In the future we may look at tentative flattening based on profiling data. But first, here are the times for repeatedly allocating an array of arrays, and for repeatedly folding the values in an array of arrays.

Next we look at allocating arrays of tuples and vice versa. We look at both square sizes and at highly skewed dimensions. This is limited by not being able to make tuples very large.
Then we do examine folding over arrays of tuples and tuples of arrays.

A Appendix: Raw numbers for above graphs

Microbenchmarks

## User time for each benchmark/backend
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2564.000 2568.000 2520.000 2524.000 2528.000 5064.000 5020.000
readfile_bigwins 3620.000 508.000 1112.000 768.000 3904.000 256.000 1020.000
printing_lists 2520.000 904.000 916.000 860.000 900.000 844.000 864.000
conv_SigsegArr 2328.000 384.000 7348.000 828.000 5844.000 40.000 6424.000
fft 124.000 868.000 964.000 924.000 1028.000 1008.000 880.000

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4580.000

Application Benchmarks:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist

## Running orig marmot phase 1
B Appendix: Additional system information

Top results before running benchmarks:

```
top - 16:56:50 up 30 days, 1:30, 7 users,  load average: 1.29, 1.19, 0.88
Tasks: 178 total, 1 running, 177 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.5%us, 4.1%sy, 1.3%ni, 66.2%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1185504k used, 888452k free, 165128k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 583492k cached

PID USER  PR  NI  VIRT  RES   SHR  S %CPU %MEM    TIME+  COMMAND
6610 newton  15   0 31596  23m  4048  S  2.0  1.1 239:33.28  unison-2.18
11719 newton  21   0 2492  1096 784  R  2.1  0.0 0:00.01  top
  root    15   0 2948  1856  532  S  0.0  0.1 0:04.69  init
  2 root   11  -5   0   0   0  S  0.0  0.0 0:00.00  kthread
  3 root  RT  -5   0   0   0  S  0.0  0.0 0:00.28  migration/0
  4 root  34   19  0  0  0  S  0.0  0.0 0:00.62  ksoftirqd/0
  5 root  RT  -5   0   0   0  S  0.0  0.0 0:00.00  watchdog/0
  6 root  RT  -5   0   0   0  S  0.0  0.0 0:00.22  migration/1
  7 root  34   19  0  0  0  S  0.0  0.0 0:01.05  ksoftirqd/1
  8 root  RT  -5   0   0   0  S  0.0  0.0 0:00.00  watchdog/1
  9 root   10  -5   0   0   0  S  0.0  0.0 0:00.03  events/0
 10 root   10  -5   0   0   0  S  0.0  0.0 0:00.04  events/1
 11 root   10  -5   0   0   0  S  0.0  0.0 0:00.02  khelper
```

Top results after running benchmarks:

```
top - 17:15:49 up 30 days, 1:49, 7 users,  load average: 1.29, 1.25, 1.10
Tasks: 178 total, 1 running, 177 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.5%us, 4.1%sy, 1.3%ni, 66.2%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1045524k used, 1028432k free, 156684k buffers
Swap: 14996668k total, 14961924k free, 583492k cached

PID USER  PR  NI  VIRT  RES   SHR  S %CPU %MEM    TIME+  COMMAND
6610 newton  15   0 31596  23m  4048  S  12.1  1.1 241:43.47  unison-2.18
21712 newton  20   0 2492  1092 784  R  2.1  0.0 0:00.01  top
  root    15   0 2948  1856  532  S  0.0  0.1 0:04.70  init
  2 root   11  -5   0   0   0  S  0.0  0.0 0:00.00  kthread
  3 root  RT  -5   0   0   0  S  0.0  0.0 0:00.28  migration/0
  4 root  34   19  0  0  0  S  0.0  0.0 0:00.62  ksoftirqd/0
  5 root  RT  -5   0   0   0  S  0.0  0.0 0:00.00  watchdog/0
  6 root  RT  -5   0   0   0  S  0.0  0.0 0:00.22  migration/1
  7 root  34   19  0  0  0  S  0.0  0.0 0:01.06  ksoftirqd/1
  8 root  RT  -5   0   0   0  S  0.0  0.0 0:00.00  watchdog/1
  9 root   10  -5   0   0   0  S  0.0  0.0 0:00.03  events/0
 10 root   10  -5   0   0   0  S  0.0  0.0 0:00.04  events/1
 11 root   10  -5   0   0   0  S  0.0  0.0 0:00.02  khelper
```