WaveScript Benchmarks Performance Report

January 2, 2008

Machine information:
Linux faith 2.6.18-4-k7 #1 SMP Wed May 9 23:42:01 UTC 2007 i686 GNU/Linux

WaveScript SVN:
Revision: 2977

WaveScope Engine SVN:
Revision: 1495

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.

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 "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFit DF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"

readfile_bigwins 2460 1236 292.0 4.0 4.0 8.0 512.0 448.0
printing_lists 9244 1768 696.0 664.0 676.0 608.0 1064.0 636.0
conv_SigsegArr 26033 64 1000.0 976.0 868.0 860.0 4.0 0.0
fft 4668 2648 400.0 316.0 372.0 360.0 24.0 40.0

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFit DF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"

fannkuch2 15653 12501 732.0 728.0 740.0 732.0 968.0 712.0

Application Benchmarks:

## Running orig marmot phase 1
run_first_phase 28846 10885 4296.0 1904.0 1764.0 1516.0 392.0 324.0
## Running marmot2
test_marmot2 13505 7524 728.0 708.0 724.0 808.0 496.0 492.0
## Running marmot3
test_heatmap 12648 7700 3092.0 3200.0 3164.0 2868.0 2344.0 2384.0

B Appendix: Additional system information

Top results before running benchmarks:

top - 09:27:17 up 156 days, 20:00, 8 users, load average: 2.15, 2.39, 2.75
Tasks: 174 total, 2 running, 165 sleeping, 6 stopped, 1 zombie
Cpu(s): 14.0%us, 1.7%sy, 2.0%ni, 80.8%id, 1.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1466576k used, 609848k free, 18032k buffers
Swap: 1951856k total, 1466576k used, 609848k free, 18032k buffers

PID USER PR NI VIRT RES SHR %CPU %MEM TIME+ COMMAND
16909 newton 25 0 9532 8176 556 R 101 0.4 77:02.05 larceny.bin
1 root 15 0 2092 88 60 S 0 0.0 2:07.90 init
2 root 97 0 0 0 0 S 0 0.0 0:05.04 migration/0
3 root 34 19 0 0 0 S 0 0.0 0:40.10 ksoftirqd/0
Top results after running benchmarks:

```
  PID   USER      PR  NI  VIRT  RES  SHR  S%CPU %MEM TIME+ COMMAND
   16909 newton  25   0 9532  8176  556  R 0.4 107:24.92 larceny.bin
      1 root     20  0  2092   88   60  S 0.0  2:07.90 init
      2 root RT  0  0  0  0 S 0 0.0 0:05.05 migration/0
      3 root 34 19  0  0  0 S 0 0.0 1:59.59 ksoftirqd/1
      4 root RT  0  0  0  0 S 0 0.0 0:05.97 migration/1
      5 root 34 19  0  0  0 S 0 0.0 0:00.16 events/0
      6 root 10 -5  0  0  0 S 0 0.0 0:00.12 events/1
      7 root 14 -5  0  0  0 S 0 0.0 0:00.00 khelper
      8 root 16 -5  0  0  0 S 0 0.0 0:00.02 kthread
     13 root 10 -5  0  0  0 S 0 0.0 0:00.94 kblockd/0
     14 root 10 -5  0  0  0 S 0 0.0 0:01.44 kblockd/1
     15 root 15 -5  0  0  0 S 0 0.0 0:00.00 kacpid
```