WaveScript Benchmarks Performance Report

September 4, 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: 3556

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

```plaintext
## User time for each benchmark/backend
Benchmark mltonO3 c2boehm c2boehmsseglist c2 c2seglist c2def c2defseglist
just_timer 2488.000 2556.000 2540.000 2528.000 2516.000 5044.000 5040.000
readfile_bigwins 3768.000 516.000 1080.000 744.000 3896.000 280.000 1000.000
printing_lists 2476.000 928.000 916.000 840.000 844.000 804.000 808.000
conv_SigsegArr 2292.000 352.000 7572.000 848.000 5636.000 40.000 6636.000
fft 124.000 908.000 948.000 868.000 904.000 872.000 856.000
```

### Language Shootout:

```plaintext
## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4404.000
```

### Application Benchmarks:

```plaintext
Benchmark mltonO3 c2boehm c2boehmsseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
```
run_first_phase 7472.000 12385.000 4268.000 7760.000 5700.000 8393.000 3880.000
## Running marmot2
test_marmot2 2308.000 5232.000 5216.000 4832.000 4680.000 4632.000 4564.000
## Running marmot3
test_heatmap 7760.000 3228.000 3188.000 2548.000 2516.000 3204.000 3192.000
## Running marmot multinode offline
run_3phases 9717.000 6212.000 4940.000 5720.000 5340.000 5728.000 4804.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 06:27:44 up 44 days, 15:01, 7 users, load average: 2.51, 2.58, 2.39
Tasks: 187 total, 2 running, 185 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.7%us, 4.4%sy, 0.9%ni, 67.1%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 551212k free, 138320k buffers
Swap: 14996668k total, 14961924k free, 34744k cached

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
7118 newton 25 0 34240 31m 492 R 98 1.6 1235:37 ikarus
30078 newton 15 0 31892 23m 4048 S 18 1.2 279:49.81 unison-2.18
6353 root 15 0 0 0 0 S 2 0.0 373:03.09 afs_rxlistener
  1 root 21 0 2948 1856 532 S 0 0.1 0:05.38 init
  2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
  3 root RT -5 0 0 0 S 0 0.0 0:00.41 migration/0
  4 root 34 19 0 0 0 S 0 0.0 0:00.79 ksoftirqd/0
  5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
  6 root RT -5 0 0 0 S 0 0.0 0:00.33 migration/1
  7 root 34 19 0 0 0 S 0 0.0 0:00.00 watchdog/1
  8 root RT -5 0 0 0 S 0 0.0 0:00.03 events/0
  9 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
```

Top results after running benchmarks:

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
7118 newton 25 0 34216 31m 492 R 97 1.6 1254:55 ikarus
  1 root 17 0 2948 1856 532 S 0 0.1 0:05.39 init
  2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
  3 root RT -5 0 0 0 S 0 0.0 0:00.41 migration/0
  4 root 34 19 0 0 0 S 0 0.0 0:00.79 ksoftirqd/0
  5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
  6 root RT -5 0 0 0 S 0 0.0 0:00.33 migration/1
  7 root 34 19 0 0 0 S 0 0.0 0:01.18 ksoftirqd/1
  8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
  9 root 10 -5 0 0 0 S 0 0.0 0:00.03 events/0
 10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
 31 root 10 -5 0 0 0 S 0 0.0 0:00.52 kblockd/0
```