WaveScript Benchmarks Performance Report

August 25, 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: 3533

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.

### Appendix: Raw numbers for above graphs

#### Microbenchmarks

```bash
## User time for each benchmark/backend
Benchmark mltonO3 c2boehm c2boehmsseglist c2 c2seglist c2def c2defseglist
just_timer 2508.000 2524.000 2520.000 2576.000 2528.000 4988.000 5044.000
readfile_bigwins 3764.000 524.000 1048.000 1336.000 3924.000 260.000 948.000
printing_lists 2696.000 904.000 900.000 880.000 852.000 824.000 824.000
conv_SigsegArr 2252.000 412.000 7540.000 852.000 5632.000 56.000 6604.000
fft 104.000 944.000 916.000 890.000 852.000 824.000 824.000
```

#### Language Shootout:

```bash
## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4464.000
```

#### Application Benchmarks:

```bash
Benchmark mltonO3 c2boehm c2boehmsseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
```
run_first_phase 7536.000 12649.000 4460.000 7756.000 5676.000 8305.000 3892.000
## Running marmot2
test_marmot2 2220.000 5240.000 5388.000 4632.000 4748.000 4616.000 4544.000
## Running marmot3
test_heatmap 7816.000 3252.000 3264.000 2548.000 2568.000 3276.000 3232.000
## Running marmot multinode offline
run_3phases 9805.000 6164.000 4872.000 5652.000 5356.000 5756.000 4876.000

B Appendix: Additional system information

Top results before running benchmarks:

```
top - 10:43:47 up 34 days, 19:17, 5 users, load average: 3.77, 3.53, 3.51
Tasks: 177 total, 3 running, 174 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.5%us, 4.1%sy, 1.2%ni, 69.3%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 829008k used, 1244948k free, 20744k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 361452k cached
```
```
PID USER PR NI VIRT RES SHARE %CPU %MEM TIME+ COMMAND
24776 newton 25 0 34224 31m 492 R 84 1.6 227:24.77 ikarus
17089 newton 25 0 41140 38m 468 R 79 1.9 0:00.64 ikarus
25064 newton 15 0 24984 16m 4044 S 13 0.8 11:05.82 unison-2.18
6353 root 15 0 2948 1856 532 S 0 0.1 0:04.98 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
3 root RT -5 0 0 0 S 0 0.0 0:00.38 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.67 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 0:00.27 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.13 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
```

Top results after running benchmarks:

```
top - 11:13:37 up 34 days, 19:47, 5 users, load average: 1.71, 2.31, 2.88
Tasks: 161 total, 2 running, 159 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.5%us, 4.1%sy, 1.2%ni, 69.3%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 708300k used, 1365656k free, 29044k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 350296k cached
```
```
PID USER PR NI VIRT RES SHARE %CPU %MEM TIME+ COMMAND
1 root 15 0 2948 1856 532 S 0 0.1 0:04.91 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
3 root RT -5 0 0 0 S 0 0.0 0:00.33 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.67 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 0:00.27 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.13 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
11 root 10 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:00.41 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1
```