WaveScript Benchmarks Performance Report

August 9, 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: 3460

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 132.000 128.000 132.000 124.000 124.000 256.000 252.000
readfile_bigwins 2176.000 236.000 240.000 4.000 8.000 20.000 16.000
printing_lists 1276.000 460.000 464.000 440.000 424.000 416.000 400.000
conv_SigsegArr 2264.000 348.000 7560.000 820.000 6060.000 36.000 6388.000
fft 64.000 476.000 516.000 460.000 448.000 492.000 456.000

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4548.000

Application Benchmarks:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
## Running marmot2

```
run_first_phase 1500.000 2488.000 852.000 1536.000 1124.000 1688.000 772.000
```

## Running marmot3

```
test_marmot2 2316.000 5244.000 5240.000 4648.000 4720.000 4588.000 4580.000
```

## Running marmot heatmap

```
test_heatmap 7848.000 3280.000 3280.000 2632.000 2632.000 3284.000 3292.000
```

## Running marmot multinode offline

```
run_3phases 9345.000 6116.000 5000.000 5716.000 5336.000 5864.000 4888.000
```

### B Appendix: Additional system information

**Top results before running benchmarks:**

```
top - 08:27:01 up 18 days, 17:01, 5 users, load average: 3.21, 3.33, 2.82
Tasks: 164 total, 3 running, 161 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.2%us, 5.1%sy, 2.2%ni, 55.5%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1396200k used, 677756k free, 108220k buffers
Swap: 14996668k total, 347440k used, 14961924k free, 700352k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
7576 newton 25 0 34212 31m 492 R 87 1.6 112:51.77 ikarus
20950 newton 25 0 29340 24m 2080 R 51 1.2 0:00.30 mzscheme
20934 newton 23 0 43120 40m 504 S 31 2.0 0:01.02 ikarus
1  root 15  0 2948 1856  532 S   0 0.1  0:03.96 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  0:00.16 migration/0
  4 root  34  19  0  0  0 S  0 0.0  0:00.34 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  0:00.12 migration/1
  7 root  34  19  0  0  0 S  0 0.0  0:00.77 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  0:00.01 events/0
 10 root  10  -5  0  0  0 S  0 0.0  0:00.03 events/1
```

**Top results after running benchmarks:**

```
top - 08:54:12 up 18 days, 17:28, 5 users, load average: 3.10, 3.14, 3.14
Tasks: 166 total, 3 running, 163 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.3%us, 5.1%sy, 2.2%ni, 55.5%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1354060k used, 720466k free, 11408k buffers
Swap: 14996668k total, 347440k used, 14961924k free, 834072k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
6935 newton 25 0 43080 40m 484 R 99 2.0 0:01.00 ikarus
7576 newton 25 0 34224 31m 492 R 98 1.6 133:44.55 ikarus
1  root 16  0 2948 1856  532 S  0 0.1  0:03.98 init
  2 root 14  -5  0  0  0 S  0 0.0  0:00.00 kthread
  3 root  RT -5  0  0  0 S  0 0.0  0:00.16 migration/0
  4 root  34  19  0  0  0 S  0 0.0  0:00.34 ksoftirqd/0
  5 root  RT -5  0  0  0 S  0 0.0  0:00.12 migration/1
  6 root  RT -5  0  0  0 S  0 0.0  0:00.12 migration/1
  7 root  34  19  0  0  0 S  0 0.0  0:00.77 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  0:00.01 events/0
 10 root  10  -5  0  0  0 S  0 0.0  0:00.03 events/1
 11 root  10  -5  0  0  0 S  0 0.0  0:00.00 khelper
```