WaveScript Benchmarks Performance Report

August 11, 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: 3464

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.

![Graph](image)

## 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 back-ends. 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 188.000 132.000 180.000 152.000 160.000 272.000 252.000
readfile_bigwins 2280.000 256.000 248.000 4.000 12.000 12.000 4.000
printing_lists 1384.000 452.000 444.000 448.000 452.000 420.000 412.000
conv_SigsegArr 2272.000 372.000 7316.000 828.000 6384.000 60.000 6496.000
fft 92.000 456.000 500.000 488.000 504.000 524.000 504.000

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4544.000

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 - 10:58:11 up 20 days, 19:32, 6 users, load average: 0.97, 1.06, 1.04
Tasks: 153 total, 2 running, 151 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.5%us, 4.7%sy, 1.9%ni, 55.9%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1438836k used, 635120k free, 144160k buffers
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 15 0 2948 1856 532 S 0.1 0:04.05 init
2 root 14 -5 0 0 0 S 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0.0 0:00.17 migration/0
4 root 34 19 0 0 0 R 0.0 0:00.45 ksoftirqd/0
5 root RT -5 0 0 0 S 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0.0 0:00.12 migration/1
7 root 39 19 0 0 0 S 0.0 0:00.84 ksoftirqd/1
8 root RT -5 0 0 0 S 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0.0 0:00.03 events/0
10 root 10 -5 0 0 0 S 0.0 0:00.03 events/1
11 root 10 -5 0 0 0 S 0.0 0:00.00 khelper
31 root 10 -5 0 0 0 S 0.0 0:00.22 kblockd/0
32 root 10 -5 0 0 0 S 0.0 0:00.00 kblockd/1
```

Top results after running benchmarks:

```
Top - 11:14:39 up 20 days, 19:48, 6 users, load average: 1.02, 1.03, 1.02
Tasks: 153 total, 1 running, 152 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.5%us, 4.7%sy, 1.9%ni, 55.9%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1401356k used, 672600k free, 41584k buffers
Swap: 1499668k total, 34744k used, 14961924k free, 776056k cached
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
8382 newton 20 0 2368 1080 784 R 2.0 0:00.01 top
1 root 18 0 2948 1856 532 S 0.1 0:04.06 init
2 root 14 -5 0 0 0 S 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0.0 0:00.17 migration/0
4 root 34 19 0 0 0 R 0.0 0:00.45 ksoftirqd/0
5 root RT -5 0 0 0 S 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0.0 0:00.12 migration/1
7 root 34 19 0 0 0 S 0.0 0:00.84 ksoftirqd/1
8 root RT -5 0 0 0 S 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0.0 0:00.03 events/0
10 root 10 -5 0 0 0 S 0.0 0:00.03 events/1
11 root 10 -5 0 0 0 S 0.0 0:00.00 khelper
31 root 10 -5 0 0 0 S 0.0 0:00.22 kblockd/0
32 root 10 -5 0 0 0 S 0.0 0:00.00 kblockd/1
```