WaveScript Benchmarks Performance Report

August 15, 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: 3491

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.
• 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 2580.000 2560.000 2532.000 2548.000 2552.000 5056.000 5072.000
readfile_bigwins 3640.000 484.000 1100.000 776.000 3932.000 288.000 1016.000
printing_lists 2644.000 912.000 936.000 844.000 828.000 868.000 840.000
conv_SigsegArr 2328.000 404.000 7308.000 812.000 6340.000 48.000 6420.000
fft 112.000 952.000 968.000 840.000 988.000 928.000 948.000

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4532.000

Application Benchmarks:

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

```
run_first_phase 1496.000 2384.000 868.000 1784.000 1164.000 1480.000 828.000
```

## Running marmot3

```
test_heatmap 7824.000 3308.000 3288.000 2632.000 2608.000 3280.000 3308.000
```

## Running marmot multinode offline

```
run_3phases 9261.000 5972.000 4980.000 5660.000 5372.000 5576.000 4888.000
```

### B Appendix: Additional system information

#### Top results before running benchmarks:

```
top - 13:32:25 up 24 days, 22:06,  7 users, load average: 0.99, 1.06, 0.93
Tasks: 168 total,  1 running, 167 sleeping,  0 stopped,  0 zombie
Cpu(s):  32.7%us,  4.2%sy,  1.6%ni, 60.6%id,  0.1%wa,  0.3%hi,  0.4%si,  0.0%st
Mem:  2073956k total,  1743604k used,  330352k free,  6916k buffers
Swap: 14996668k total,  34756k used, 14961912k free,  1424100k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 15  0  2948  1856  532 S  0.1  0:04.56 init
2 root 11 -5  0  0  0  S  0.0  0:00.00 kthreadd
3 root RT -5  0  0  0  S  0.0  0:00.25 migration/0
4 root 34 19  0  0  0  S  0.0  0:00.58 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  0:00.19 migration/1
7 root 34 19  0  0  0  S  0.0  0:01.00 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  0:00.04 events/1
11 root 10 -5  0  0  0  S  0.0  0:00.02 khelper
31 root 10 -5  0  0  0  S  0.0  0:00.28 kblockd/0
32 root 10 -5  0  0  0  S  0.0  0:00.00 kblockd/1
```

#### Top results after running benchmarks:

```
top - 13:49:46 up 24 days, 22:23,  7 users, load average: 1.01, 1.06, 1.00
Tasks: 168 total,  1 running, 167 sleeping,  0 stopped,  0 zombie
Cpu(s):  32.7%us,  4.2%sy,  1.6%ni, 60.6%id,  0.1%wa,  0.3%hi,  0.4%si,  0.0%st
Mem:  2073956k total,  1076332k used,  997624k free,  8224k buffers
Swap: 14996668k total,  34756k used, 14961912k free,  1424100k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
21906 newton 21  0  2368  1088  784 R  2.1  0:00.01 top
1 root 15  0  2948  1856  532 S  0.1  0:04.56 init
2 root 11 -5  0  0  0  S  0.0  0:00.00 kthreadd
3 root RT -5  0  0  0  S  0.0  0:00.25 migration/0
4 root 34 19  0  0  0  S  0.0  0:00.58 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  0:00.20 migration/1
7 root 34 19  0  0  0  S  0.0  0:01.00 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  0:00.04 events/1
11 root 10 -5  0  0  0  S  0.0  0:00.02 khelper
31 root 10 -5  0  0  0  S  0.0  0:00.28 kblockd/0
32 root 10 -5  0  0  0  S  0.0  0:00.00 kblockd/1
```