WaveScript Benchmarks Performance Report

August 5, 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: 3430

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 c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 124.000 168.000 140.000 148.000 128.000 252.000 252.000
readfile_bigwins 748.000 64.000 76.000 0 4.000 12.000 0
printing_lists 1352.000 456.000 452.000 424.000 420.000 408.000 412.000
conv_SigsegArr 556.000 108.000 1816.000 244.000 1392.000 24.000 1576.000
fft 88.000 496.000 460.000 460.000 484.000 488.000 472.000
```

Language Shootout:

```plaintext
## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4536.000
```

Application Benchmarks:

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

test_marmot2 2212.000 5260.000 5272.000 4656.000 4708.000 4564.000 4564.000

## Running marmot3

test_heatmap 7856.000 3232.000 3240.000 2588.000 2584.000 3228.000 3236.000

## Running marmot multinode offline

run_3phases 9209.000 5888.000 4860.000 5568.000 5256.000 5508.000 4804.000

### Appendix: Additional system information

Top results before running benchmarks:

```
run - 06:13:51 up 14 days, 14:48, 5 users, load average: 0.93, 1.00, 0.66
Tasks: 158 total, 2 running, 156 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.6%us, 5.1%sy, 0.0%ni, 57.4%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 675700k used, 1398256k free, 95072k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 321992k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
15661 newton 20 0 2360 1076 784 R 2 0.1 0:00.01 top
1 root 18 0 2948 1856 532 S 0 0.1 0:03.46 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 0:00.11 migration/0
4 root 34 19 0 0 0 R 0 0.0 0:00.28 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 0:00.08 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:00.69 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
31 root 10 -5 0 0 0 S 0 0.0 0:00.15 kblockd/0
```

Top results after running benchmarks:

```
run - 06:30:05 up 14 days, 15:04, 5 users, load average: 1.18, 1.17, 0.98
Tasks: 158 total, 1 running, 157 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.6%us, 5.1%sy, 0.0%ni, 57.4%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 816588k used, 1257368k free, 97524k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 458536k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
11358 newton 15 0 26384 17a 4044 S 18 0.9 143:20.84 unison-2.18
6333 root 15 0 0 0 0 S 4 0.0 129:41.11 afs_rxlistener
24129 newton 21 0 2364 1080 746 R 2 0.1 0:00.01 top
1 root 18 0 2948 1856 532 S 0 0.1 0:03.48 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 0:00.11 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.28 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 0:00.08 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:00.69 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
```