WaveScript Benchmarks Performance Report

September 26, 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: 3572

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.

![fannkuch2](image_url)

## 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

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2540.000 2516.000 2524.000 2528.000 2508.000 5032.000 5036.000
readfile_bigwins 3788.000 504.000 1128.000 792.000 3868.000 268.000 996.000
printing_lists 2724.000 900.000 960.000 872.000 844.000 816.000 996.000
conv_SigsegArr 2320.000 352.000 7464.000 836.000 5696.000 68.000 6724.000
fft 116.000 884.000 920.000 880.000 904.000 908.000 864.000
```

Language Shootout:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4468.000
```
## Application Benchmarks:

### Real or User time for each benchmark/backend

### LD_PRELOAD:

### NOSUDO:

### NICE:

Benchmark mltonO3 c2boehm c2boehmsglist c2 c2seglist c2def c2defseglist

### Running orig marmot phase 1

run_first_phase 7488.000 12353.000 4288.000 7712.000 5676.000 8365.000 3904.000

### Running marmot2

test_marmot2 2324.000 5256.000 5248.000 4708.000 4684.000 4564.000 4544.000

### Running marmot3

test_heatmap 7768.000 3208.000 3212.000 2520.000 2516.000 3200.000 3220.000

### Running marmot multinode offline

run_3phases 9601.000 6112.000 5620.000 35688k used, 35688k buffers

## Appendix: Additional system information

### Top results before running benchmarks:

```
top - 06:22:41 up 66 days, 14:56,  6 users,  load average: 1.94, 1.99, 1.75
Tasks: 183 total,  2 running, 181 sleeping,  0 stopped,  0 zombie
Cpu(s):  32.0%us,  3.8%sy,  1.0%ni, 62.4%id,  0.2%wa,  0.3%hi,  0.3%si,  0.0%st
Mem: 2073956k total, 1582928k used,  491028k free, 35688k buffers
Swap: 14996668k total, 35204k used, 14961464k free, 1113816k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
27326 newton 39 19 34212 31m 492 R 99 1.6 12820:16 ikarus
  1 root 18  0 2948 1856 532 S  0 0.1 0:06.81 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 0:03.88 migration/0
  4 root 34 19  0  0  0 S  0 0.0 0:14.29 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 0:03.28 migration/1
  7 root 34 19  0  0  0 S  0 0.0 0:03.85 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 16 -5  0  0  0 S  0 0.0 0:00.02 khelper
 31 root 10 -5  0  0  0 S  0 0.0 0:01.01 kblockd/0

```

### Top results after running benchmarks:

```
top - 06:41:42 up 66 days, 15:15,  6 users,  load average: 2.01, 2.03, 1.93
Tasks: 183 total,  2 running, 181 sleeping,  0 stopped,  0 zombie
Cpu(s):  32.0%us,  3.8%sy,  1.0%ni, 62.4%id,  0.2%wa,  0.3%hi,  0.3%si,  0.0%st
Mem: 2073956k total, 1047500k used, 1026456k free, 23596k buffers
Swap: 14996668k total, 35204k used, 14961464k free, 596572k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
27326 newton 39 19 34248 31m 492 R 99 1.6 12820:59 ikarus
  1 root 18  0 2948 1856 532 S  0 0.1 0:06.81 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 0:03.88 migration/0
  4 root 34 19  0  0  0 S  0 0.0 0:14.29 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 0:03.28 migration/1
  7 root 34 19  0  0  0 S  0 0.0 0:03.85 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 16 -5  0  0  0 S  0 0.0 0:00.02 khelper
 31 root 10 -5  0  0  0 S  0 0.0 0:01.01 kblockd/0

```