WaveScript Benchmarks Performance Report

October 31, 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: 3621

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.

3.1 Marmot Application

We start off by looking at the original, hand-optimized marmot application that we deployed. We break it down by phase: the first three phases of the computation, followed by all three together.
3.2 Computer Vision: Background Subtraction

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.
## Appendix: Raw numbers for above graphs

### Microbenchmarks

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

**LD_PRELOAD:**

**NOSUDO:**

**NICE:**

Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist

just_timer 2536.000 2532.000 2528.000 2528.000 2512.000 5024.000 5036.000

readfile_bigwins 3736.000 472.000 1220.000 1264.000 3756.000 252.000 876.000

printing_lists 2644.000 896.000 892.000 880.000 888.000 780.000 828.000

conv_SigsegArr 2352.000 416.000 7280.000 828.000 5704.000 48.000 6576.000

fft 104.000 956.000 924.000 896.000 1028.000 844.000 976.000

### Language Shootout

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

**LD_PRELOAD:**

**NOSUDO:**

**NICE:**

Benchmark c2

fannkuch2 4328.000

### Application Benchmarks

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

**LD_PRELOAD:**

**NOSUDO:**

**NICE:**

Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist

Running benchmark marmot1.bench for 100 tuples.

run_first_phase 7268.000 11717.000 4120.000 7372.000 5512.000 7364.000 3916.000

Running benchmark marmot2.bench for 150 tuples.

test_marmot2 2212.000 5244.000 5256.000 4652.000 4736.000 4540.000 4552.000

Running benchmark marmot3.bench for 14 tuples.

test_heatmap 7744.000 3256.000 3224.000 2508.000 2508.000 3248.000 3240.000

Running benchmark marmot_all.bench for 20 tuples.
un_3phases 9497.000 5864.000 4816.000 5624.000 5312.000 5512.000 4812.000

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

**LD_PRELOAD:**

**NOSUDO:**
## NICE:
Benchmark mltonO3 c2boehm c2 c2def
Running benchmark bgsusb3.bench for 10 tuples.
bgSub3_integer 10217.000 8345.000 9821.000 7688.000

### B Appendix: Additional system information

Top results before running benchmarks:

```
top - 11:08:46 up 101 days, 19:42, 5 users, load average: 1.07, 1.02, 0.77
Tasks: 195 total, 1 running, 194 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.2%us, 2.8%sy, 0.8%ni, 70.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1443216k used, 630740k free, 54828k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 1019624k cached
```

```
   PID USER PR NI  VIRT  RES  SHR S %CPU %MEM TIME+ COMMAND
  1 root  21   0 2948  1856  532 S  0  0.1 0:07.98  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 0:05.57  migration/0
  4 root  34  19   0   0   0 S  0  0.0 0:14.75  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 0:04.77  migration/1
  7 root  34  19   0   0   0 S  0  0.0 0:03.98  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 0:01.78  events/0
 10 root  10  -5   0   0   0 S  0  0.0 0:00.04  events/1
 11 root  10  -5   0   0   0 S  0  0.0 0:00.02  khelper
 31 root  10  -5   0   0   0 S  0  0.0 0:01.64  kblockd/0
 32 root  10  -5   0   0   0 S  0  0.0 0:00.04  kblockd/1
```

Top results after running benchmarks:

```
top - 11:29:02 up 101 days, 20:03, 5 users, load average: 0.98, 1.00, 0.92
Tasks: 195 total, 1 running, 194 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.2%us, 2.8%sy, 0.8%ni, 70.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1143216k used, 924740k free, 54828k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 739352k cached
```

```
   PID USER PR NI  VIRT  RES  SHR S %CPU %MEM TIME+ COMMAND
  1 root  15   0 2948  1856  532 S  0  0.1 0:07.99  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 0:05.58  migration/0
  4 root  34  19   0   0   0 S  0  0.0 0:14.75  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 0:04.77  migration/1
  7 root  34  19   0   0   0 S  0  0.0 0:03.98  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 0:01.78  events/0
 10 root  10  -5   0   0   0 S  0  0.0 0:00.04  events/1
 11 root  10  -5   0   0   0 S  0  0.0 0:00.02  khelper
 31 root  10  -5   0   0   0 S  0  0.0 0:01.64  kblockd/0
 32 root  10  -5   0   0   0 S  0  0.0 0:00.04  kblockd/1
```