WaveScript Benchmarks Performance Report

August 6, 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: 3433

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 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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mltonO3</th>
<th>c2boehm</th>
<th>c2boehmselist</th>
<th>c2</th>
<th>c2seglist</th>
<th>c2def</th>
<th>c2defseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td>just_timer</td>
<td>132.000</td>
<td>128.000</td>
<td>124.000</td>
<td>128.000</td>
<td>124.000</td>
<td>252.000</td>
<td>256.000</td>
</tr>
<tr>
<td>readfile_bigwins</td>
<td>564.000</td>
<td>60.000</td>
<td>64.000</td>
<td>0</td>
<td>0</td>
<td>12.000</td>
<td>0</td>
</tr>
<tr>
<td>printing_lists</td>
<td>1392.000</td>
<td>448.000</td>
<td>444.000</td>
<td>432.000</td>
<td>460.000</td>
<td>416.000</td>
<td>424.000</td>
</tr>
<tr>
<td>conv_SigsegArr</td>
<td>556.000</td>
<td>84.000</td>
<td>1812.000</td>
<td>188.000</td>
<td>1612.000</td>
<td>4.000</td>
<td>1616.000</td>
</tr>
<tr>
<td>fft</td>
<td>56.000</td>
<td>476.000</td>
<td>520.000</td>
<td>448.000</td>
<td>496.000</td>
<td>416.000</td>
<td>460.000</td>
</tr>
</tbody>
</table>

Language Shootout:

## User time for each language-shootout benchmark/backend

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>fannkuch2</td>
<td>4492.000</td>
</tr>
</tbody>
</table>

Application Benchmarks:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mltonO3</th>
<th>c2boehm</th>
<th>c2boehmselist</th>
<th>c2</th>
<th>c2seglist</th>
<th>c2def</th>
<th>c2defseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td>just_timer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>readfile_bigwins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>printing_lists</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv_SigsegArr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Running orig marmot phase 1
B Appendix: Additional system information

Top results before running benchmarks:

```
top - 11:36:19 up 15 days, 20:10, 5 users, load average: 2.08, 2.10, 1.76
Tasks: 162 total, 2 running, 160 sleeping, 0 stopped, 0 zombie
Cpu(s): 37.7%us, 5.1%sy, 0.0%ni, 56.1%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1541532k used, 532424k free, 118936k buffers
Swap: 14996668k total, 34740k used, 14961928k free, 993292k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
24183 newton 25 0 34212 31m 492 R 100 1.6 1739:55 ikarus
  1 root  18  0 2948 1856 532 S  0.1 0:03:59 init
  2 root  11 -5  0  0  0 S  0.0 0:00.00 kthread
  3 root RT -5  0  0  0 S  0.0 0:00.13 migration/0
  4 root  39 19  0  0  0 S  0.0 0:00.30 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 0:00.08 migration/1
  7 root  34 19  0  0  0 S  0.0 0:00.71 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 0:00.01 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 0:00.17 kblockd/0
```

Top results after running benchmarks:

```
top - 11:54:10 up 15 days, 20:28, 5 users, load average: 2.16, 2.15, 2.04
Tasks: 163 total, 2 running, 161 sleeping, 0 stopped, 0 zombie
Cpu(s): 37.8%us, 5.1%sy, 0.0%ni, 56.1%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1396848k used, 677108k free, 119792k buffers
Swap: 14996668k total, 34740k used, 14961928k free, 850156k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
24183 newton 25 0 34212 31m 492 R 100 1.6 1756:52 ikarus
  1 root  18  0 2948 1856 532 S  0.1 0:03:60 init
  2 root  11 -5  0  0  0 S  0.0 0:00.00 kthread
  3 root RT -5  0  0  0 S  0.0 0:00.13 migration/0
  4 root  39 19  0  0  0 S  0.0 0:00.30 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 0:00.08 migration/1
  7 root  34 19  0  0  0 S  0.0 0:00.71 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 0:00.01 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 0:00.17 kblockd/0
```