WaveScript Benchmarks Performance Report

August 29, 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: 3545

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

## User time for each benchmark/backend

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mltonO3</th>
<th>c2boehm</th>
<th>c2boehmseglist</th>
<th>c2</th>
<th>c2seglist</th>
<th>c2def</th>
<th>c2defseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td>just_timer</td>
<td>2584.000</td>
<td>2512.000</td>
<td>2516.000</td>
<td>2512.000</td>
<td>2560.000</td>
<td>4976.000</td>
<td>5064.000</td>
</tr>
<tr>
<td>readfile_bigwins</td>
<td>3792.000</td>
<td>492.000</td>
<td>1108.000</td>
<td>776.000</td>
<td>3860.000</td>
<td>284.000</td>
<td>1072.000</td>
</tr>
<tr>
<td>printing_lists</td>
<td>2986.000</td>
<td>908.000</td>
<td>916.000</td>
<td>896.000</td>
<td>865.000</td>
<td>880.000</td>
<td>880.000</td>
</tr>
<tr>
<td>conv_SigsegArr</td>
<td>2272.000</td>
<td>428.000</td>
<td>7328.000</td>
<td>784.000</td>
<td>5540.000</td>
<td>36.000</td>
<td>6616.000</td>
</tr>
<tr>
<td>fft</td>
<td>120.000</td>
<td>940.000</td>
<td>904.000</td>
<td>812.000</td>
<td>940.000</td>
<td>880.000</td>
<td>828.000</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>4484.000</td>
</tr>
</tbody>
</table>

Application Benchmarks:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist

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

Top results before running benchmarks:

```
top - 13:45:25 up 38 days, 22:19, 6 users, load average: 1.04, 1.15, 1.01
Tasks: 167 total, 1 running, 166 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.0%us, 4.4%sy, 1.0%ni, 68.6%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1312940k used, 761016k free, 129416k buffers
Swap: 14996668k total, 34752k used, 14961916k free, 774832k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 15 0 2948 1856 532 S 0 0.1 0:05.08 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 0:00.37 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.69 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 0:00.30 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.16 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 10 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:00.46 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1
```