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: 3436

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
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 124.000 124.000 128.000 120.000 128.000 252.000 268.000
readfile_bigwins 584.000 116.000 88.000 0 0 4.000 8.000
printing_lists 1404.000 460.000 456.000 440.000 424.000 416.000 408.000
conv_SigsegArr 580.000 80.000 1828.000 180.000 1536.000 0 1576.000
fft 60.000 472.000 520.000 464.000 468.000 452.000 448.000

Language Shootout:

### User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4556.000

Application Benchmarks:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
## Run First Phase

### Running marmot2

### Running marmot3

### Running marmot3 offline

---

### Appendix: Additional system information

**Top results before running benchmarks:**

```
Top - 14:06:39 up 15 days, 22:40, 6 users, load average: 3.31, 3.28, 2.56
Tasks: 172 total, 4 running, 168 sleeping, 0 stopped, 0 zombie
Cpu(s): 37.9%us, 5.2%sy, 0.0%ni, 56.0%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1264420k used, 809536k free, 95080k buffers
Swap: 14996668k total, 1264420k used, 13732248k free, 703960k cached
```

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>24183</td>
<td>newton</td>
<td>25</td>
<td>0</td>
<td>34224</td>
<td>31m</td>
<td>492</td>
<td>R</td>
<td>99</td>
<td>1.6</td>
<td>1886:05</td>
<td>ikarus</td>
</tr>
<tr>
<td>31426</td>
<td>newton</td>
<td>25</td>
<td>0</td>
<td>42884</td>
<td>35m</td>
<td>2012</td>
<td>R</td>
<td>97</td>
<td>1.8</td>
<td>00:54:52</td>
<td>mzscheme</td>
</tr>
</tbody>
</table>

```
1 root 18 0 2948 1856 532 S 0 0.1 03:03:11 init
2 root 11 -5 0 0 0 S 0 0.0 00:00:00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 00:13:13 migration/0
4 root 34 19 0 0 0 S 0 0.0 00:30:30 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 00:00:00 watchdogd/0
6 root RT -5 0 0 0 S 0 0.0 00:00:08 migration/1
7 root 34 19 0 0 0 S 0 0.0 00:00:00 ksoftirqd/1
9 root 10 -5 0 0 0 S 0 0.0 00:00:01 events/0
10 root 10 -5 0 0 0 S 0 0.0 00:00:03 events/1
11 root 10 -5 0 0 0 S 0 0.0 00:00:00 khelper
```

**Top results after running benchmarks:**

```
Top - 14:34:21 up 15 days, 23:08, 6 users, load average: 3.42, 3.46, 3.31
Tasks: 174 total, 3 running, 171 sleeping, 0 stopped, 0 zombie
Cpu(s): 37.9%us, 5.2%sy, 0.0%ni, 55.9%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1264420k used, 756460k free, 100600k buffers
Swap: 14996668k total, 131496k used, 13881600k free, 742460k cached
```

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>24183</td>
<td>newton</td>
<td>25</td>
<td>0</td>
<td>34212</td>
<td>31m</td>
<td>492</td>
<td>R</td>
<td>99</td>
<td>1.6</td>
<td>1907:11</td>
<td>ikarus</td>
</tr>
<tr>
<td>18843</td>
<td>newton</td>
<td>25</td>
<td>0</td>
<td>40584</td>
<td>38m</td>
<td>468</td>
<td>R</td>
<td>83</td>
<td>1.9</td>
<td>00:42:02</td>
<td>ikarus</td>
</tr>
</tbody>
</table>

```
1 root 18 0 2948 1856 532 S 0 0.1 03:03:03 init
2 root 11 -5 0 0 0 S 0 0.0 00:00:00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 00:13:30 migration/0
4 root 34 19 0 0 0 S 0 0.0 00:30:30 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 00:00:00 watchdogd/0
6 root RT -5 0 0 0 S 0 0.0 00:00:08 migration/1
7 root 34 19 0 0 0 S 0 0.0 00:00:00 ksoftirqd/1
9 root 10 -5 0 0 0 S 0 0.0 00:00:01 events/0
10 root 10 -5 0 0 0 S 0 0.0 00:00:03 events/1
```

---

6