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: 3546

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.
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>c2boehmsglist</th>
<th>c2</th>
<th>c2seglist</th>
<th>c2def</th>
<th>c2defseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td>just_timer</td>
<td>2544.000</td>
<td>2508.000</td>
<td>2576.000</td>
<td>2516.000</td>
<td>2548.000</td>
<td>5020.000</td>
<td>5060.000</td>
</tr>
<tr>
<td>readfile_bigwins</td>
<td>3760.000</td>
<td>500.000</td>
<td>1080.000</td>
<td>828.000</td>
<td>3924.000</td>
<td>224.000</td>
<td>1052.000</td>
</tr>
<tr>
<td>printing_lists</td>
<td>2596.000</td>
<td>924.000</td>
<td>888.000</td>
<td>864.000</td>
<td>864.000</td>
<td>864.000</td>
<td>812.000</td>
</tr>
<tr>
<td>conv_SigsegArr</td>
<td>2272.000</td>
<td>456.000</td>
<td>7372.000</td>
<td>832.000</td>
<td>5344.000</td>
<td>44.000</td>
<td>6584.000</td>
</tr>
<tr>
<td>fft</td>
<td>108.000</td>
<td>928.000</td>
<td>876.000</td>
<td>816.000</td>
<td>960.000</td>
<td>900.000</td>
<td>836.000</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>4440.000</td>
</tr>
</tbody>
</table>

Application Benchmarks:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mltonO3</th>
<th>c2boehm</th>
<th>c2boehmsglist</th>
<th>c2</th>
<th>c2seglist</th>
<th>c2def</th>
<th>c2defseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td>just_timer</td>
<td>2544.000</td>
<td>2508.000</td>
<td>2576.000</td>
<td>2516.000</td>
<td>2548.000</td>
<td>5020.000</td>
<td>5060.000</td>
</tr>
<tr>
<td>readfile_bigwins</td>
<td>3760.000</td>
<td>500.000</td>
<td>1080.000</td>
<td>828.000</td>
<td>3924.000</td>
<td>224.000</td>
<td>1052.000</td>
</tr>
<tr>
<td>printing_lists</td>
<td>2596.000</td>
<td>924.000</td>
<td>888.000</td>
<td>864.000</td>
<td>864.000</td>
<td>864.000</td>
<td>812.000</td>
</tr>
<tr>
<td>conv_SigsegArr</td>
<td>2272.000</td>
<td>456.000</td>
<td>7372.000</td>
<td>832.000</td>
<td>5344.000</td>
<td>44.000</td>
<td>6584.000</td>
</tr>
<tr>
<td>fft</td>
<td>108.000</td>
<td>928.000</td>
<td>876.000</td>
<td>816.000</td>
<td>960.000</td>
<td>900.000</td>
<td>836.000</td>
</tr>
</tbody>
</table>

## Running orig marmot phase 1
run_first_phase 7384.000 12401.000 4484.000 7840.000 5692.000 8129.000 3956.000
## Running marmot2
test_marmot2 2208.000 5280.000 5340.000 4700.000 4656.000 4540.000 4560.000
## Running marmot3
test_heatmap 7844.000 3292.000 3208.000 2520.000 2516.000 3204.000 3268.000
## Running marmot multinode offline
run_3phases 9685.000 6148.000 4820.000 5616.000 5288.000 5708.000 4876.000

B Appendix: Additional system information

Top results before running benchmarks:

```
top - 15:14:30 up 38 days, 23:48, 6 users, load average: 0.94, 1.13, 1.01
Tasks: 166 total, 1 running, 165 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.0%us, 4.4%sy, 1.0%ni, 68.5%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 893208k used, 1180748k free, 132920k buffers
Swap: 14996668k total, 34752k used, 14961916k free, 649096k cached
```

```
P ID  USER PR NI VIRT  RES   SHR S %CPU %MEM  TIME+ COMMAND
25064 newton 15  0  2948 1856  532 S   0  0.1  0:05.11 init
  2 root  15  11 -5  0   0   0 S    0  0.0  0:00.00 kthreadd
  3 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/0
  4 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/0
  6 root  15  34  19  0   0   0 S    0  0.0  0:00.30 migration/1
  7 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
  8 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
  9 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
 10 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
 11 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
 12 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
```

Top results after running benchmarks:

```
top - 15:36:31 up 39 days, 10 min, 6 users, load average: 2.46, 2.59, 2.07
Tasks: 176 total, 2 running, 174 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.0%us, 4.4%sy, 1.0%ni, 68.5%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 893208k used, 1180748k free, 132920k buffers
Swap: 14996668k total, 34752k used, 14961916k free, 649096k cached
```

```
P ID  USER PR NI VIRT  RES   SHR S %CPU %MEM  TIME+ COMMAND
490 newton 25  0  50404 47m  712 R  101 2.3 0:04.44 scheme
25064 newton 15  0  48696 39m  4048 D  14 2.0 757:11.28 unison-2.18
32643 newton 15  0  2948 1856  532 S    0  0.1  0:05.12 init
  2 root  15  11 -5  0   0   0 S    0  0.0  0:00.00 kthreadd
  3 root  15  34  19  0   0   0 S    0  0.0  0:00.30 migration/0
  4 root  15  34  19  0   0   0 S    0  0.0  0:00.37 migration/0
  6 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/0
  8 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
 10 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
 11 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
 12 root  15  34  19  0   0   0 S    0  0.0  0:00.00 watchdog/1
```