WaveScript Benchmarks Performance Report

October 3, 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: 3583

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

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2548.000 2520.000 2512.000 2516.000 2516.000 5032.000 5028.000
readfile_bigwins 3820.000 500.000 1104.000 760.000 4008.000 272.000 1028.000
printing_lists 2696.000 944.000 884.000 864.000 860.000 820.000 800.000
conv_SigsegArr 2284.000 476.000 7496.000 796.000 5768.000 56.000 6708.000
fft 124.000 868.000 1032.000 904.000 928.000 924.000 940.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4456.000
Application Benchmarks:

## Real or User time for each benchmark/backend

## LD_PRELOAD:
## NOSUDO:
## NICE:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist

## Running orig marmot phase 1
run_first_phase 7336.000 12277.000 4240.000 7924.000 5688.000 8433.000 3828.000

## Running marmot2
test_marmot2 2328.000 5248.000 5236.000 4720.000 4684.000 4568.000 4572.000

## Running marmot3
test_heatmap 7808.000 3204.000 3208.000 2512.000 2512.000 3216.000 3252.000

## Running marmot multinode offline
run_3phases 9537.000 6116.000 4908.000 5688.000 5308.000 5752.000 4796.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 15:52:07 up 74 days, 26 min, 2 users, load average: 1.92, 2.02, 1.80
Tasks: 164 total, 2 running, 162 sleeping, 0 stopped, 0 zombie
Cpu(s): 32.4%us, 3.7%sy, 1.0%ni, 62.2%id, 0.2%wa, 0.2%hi, 0.3%si, 0.0%st
Mem: 2073956k total, 1591656k used, 482300k free, 119180k buffers
Swap: 14996668k total, 35592k used, 14961076k free, 966788k 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>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>30600</td>
<td>newton</td>
<td>25</td>
<td>0</td>
<td>34220</td>
<td>31m</td>
<td>492 R</td>
<td>1.6</td>
<td>990:28</td>
<td>ikarus</td>
<td></td>
</tr>
<tr>
<td>32050</td>
<td>newton</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>1084</td>
<td>784 R</td>
<td>0.1</td>
<td>0:00:01</td>
<td>top</td>
<td></td>
</tr>
</tbody>
</table>

Top results after running benchmarks:

top - 16:11:14 up 74 days, 45 min, 3 users, load average: 2.20, 2.08, 1.97
Tasks: 168 total, 2 running, 166 sleeping, 0 stopped, 0 zombie
Cpu(s): 32.4%us, 3.7%sy, 1.0%ni, 62.2%id, 0.2%wa, 0.2%hi, 0.3%si, 0.0%st
Mem: 2073956k total, 1057768k used, 1016188k free, 116524k buffers
Swap: 14996668k total, 35592k used, 14961076k free, 474256k 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>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>30600</td>
<td>newton</td>
<td>25</td>
<td>0</td>
<td>34208</td>
<td>31m</td>
<td>492 R</td>
<td>1.6</td>
<td>2009:29</td>
<td>ikarus</td>
<td></td>
</tr>
<tr>
<td>32643</td>
<td>newton</td>
<td>15</td>
<td>0</td>
<td>15836</td>
<td>11m</td>
<td>4200 S</td>
<td>0.6</td>
<td>196:43:12</td>
<td>emacs21-x</td>
<td></td>
</tr>
<tr>
<td>1 root</td>
<td>18</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532 S</td>
<td>0.1</td>
<td>0:07:03</td>
<td>init</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 root</td>
<td>11</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:04.82</td>
<td>migration/0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:14.61</td>
<td>ksoftirqd/0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:03.95</td>
<td>migration/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 root</td>
<td>38</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:03.87</td>
<td>ksoftirqd/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:00.03</td>
<td>events/0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0 S</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Top results after running benchmarks: