WaveScript Benchmarks Performance Report

November 4, 2008

Machine information:
Linux chastity 2.6.22-14-generic #1 SMP Tue Feb 12 07:42:25 UTC 2008 i686 GNU/Linux
LD_PRELOAD: CC: cc

WaveScript SVN:
Revision: 3625

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.

3.1 Marmot Application

We start off by looking at the original, hand-optimized marmot application that we deployed. We break it down by phase: the first three phases of the computation, followed by all three together.
3.2 Computer Vision: Background Subtraction

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

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' = Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2544.000 2544.000 2512.000 2520.000 2532.000 5016.000 5032.000
readfile_bigwins 3676.000 456.000 1124.000 1292.000 3776.000 328.000 852.000
printing_lists 2684.000 900.000 904.000 848.000 892.000 812.000 832.000
conv_SigSegArr 2296.000 404.000 7288.000 848.000 5720.000 60.000 6504.000
fft 152.000 956.000 888.000 872.000 888.000 912.000 944.000
```

Language Shootout:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' = Benchmark c2
fannkuch2 4312.000
```

Application Benchmarks:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' = Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7260.000 11825.000 4068.000 7372.000 5484.000 7392.000 3980.000
## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2220.000 5268.000 5268.000 4648.000 4736.000 4536.000 4572.000
## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7800.000 3248.000 3236.000 2532.000 2516.000 3232.000 3240.000
```
## Running benchmark marmot_all.bench for 20 tuples.
run_3phases 9537.000 5884.000 4840.000 5660.000 5284.000 5492.000 4808.000

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## `which icc`

Benchmark mltonO3 c2boehm c2 c2def

## Running benchmark bgsub3.bench for 10 tuples.
bgSub3_integer 10189.000 8345.000 9869.000 7640.000

# Appendix: Additional system information

Top results before running benchmarks:
top - 06:22:49 up 105 days, 15:56, 6 users, load average: 1.05, 1.06, 0.80
Tasks: 194 total, 1 running, 193 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.4%us, 2.7%sy, 0.7%ni, 71.6%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 931548k used, 1142408k free, 47604k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 599116k 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>1</td>
<td>root</td>
<td>21</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532</td>
<td>0.1</td>
<td>0:08:23</td>
<td>init</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>11</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:14.76</td>
<td>ksoftirqd/0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchd/0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:03.98</td>
<td>ksoftirqd/1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchd/1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:01.78</td>
<td>events/0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.02</td>
<td>khelper</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:01.71</td>
<td>kblockd/0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>kblockd/1</td>
<td></td>
</tr>
</tbody>
</table>

Top results after running benchmarks:
top - 06:43:03 up 105 days, 16:17, 6 users, load average: 1.01, 1.01, 0.93
Tasks: 194 total, 1 running, 193 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.4%us, 2.7%sy, 0.7%ni, 71.6%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 933392k used, 1140564k free, 51780k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 594848k 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>30597</td>
<td>newton</td>
<td>21</td>
<td>0</td>
<td>2492</td>
<td>1096</td>
<td>784</td>
<td>2.1</td>
<td>0:00:01</td>
<td>top</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532</td>
<td>0.1</td>
<td>0:08:24</td>
<td>init</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>11</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:14.76</td>
<td>ksoftirqd/0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:03.98</td>
<td>ksoftirqd/1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchd/0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:01.78</td>
<td>events/0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.02</td>
<td>khelper</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:01.71</td>
<td>kblockd/0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>kblockd/1</td>
<td></td>
</tr>
</tbody>
</table>

7