WaveScript Benchmarks Performance Report

November 6, 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: 3633

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 back-ends. 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.
## 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 2532.000 2512.000 2516.000 2516.000 2516.000 5020.000 5020.000
readfile_bigwins 3704.000 464.000 1164.000 1296.000 3804.000 280.000 856.000
printing_lists 2696.000 924.000 888.000 848.000 872.000 816.000 824.000
conv_SigsegArr 2272.000 408.000 7336.000 844.000 6696.000 44.000 6436.000
fft 120.000 972.000 876.000 912.000 988.000 816.000 840.000
```

### Language Shootout:

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

Benchmark c2
fannkuch2 4508.000
```

### 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 7324.000 11805.000 4160.000 7364.000 5552.000 7316.000 3952.000
Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2220.000 5264.000 5264.000 4632.000 4632.000 4568.000 4572.000
Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7784.000 3268.000 3260.000 2540.000 2536.000 3252.000 3244.000
```
## Running benchmark marmot_all.bench for 20 tuples.

```
run_3phases 9517.000 5888.000 5668.000 5288.000 5516.000 4836.000
```

## Real or User time for each benchmark/backend

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

```
Benchmark mltonO3 c2boehm c2 c2def
```

## Running benchmark bgsub3.bench for 10 tuples.

```
bgSub3_integer 10213.000 8321.000 9813.000 7632.000
```

---

### Appendix: Additional system information

#### Top results before running benchmarks:

```
top - 13:18:02 up 107 days, 22:52, 6 users, load average: 1.09, 1.49, 1.46
Tasks: 193 total, 1 running, 192 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.0%us, 2.7%sy, 0.7%ni, 72.1%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1231096k used, 842860k free, 76728k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 769792k 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>8308</td>
<td>newton</td>
<td>21</td>
<td>0</td>
<td>2496</td>
<td>1100</td>
<td>784</td>
<td>R</td>
<td>2</td>
<td>0.1</td>
<td>0:00.01</td>
<td>top</td>
</tr>
<tr>
<td>32643</td>
<td>newton</td>
<td>15</td>
<td>0</td>
<td>15836</td>
<td>11m</td>
<td>4200</td>
<td>S</td>
<td>2</td>
<td>0.6</td>
<td>397:03:03</td>
<td>emacs21-x</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>21</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532</td>
<td>S</td>
<td>0</td>
<td>0.1</td>
<td>0:08.38</td>
<td>init</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:05.70</td>
<td>migration/0</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:14.79</td>
<td>ksoftirqd/0</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/0</td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:04.88</td>
<td>migration/1</td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:04.00</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/1</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:01.78</td>
<td>events/0</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>15</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.02</td>
<td>khelper</td>
</tr>
</tbody>
</table>

#### Top results after running benchmarks:

```
top - 13:37:53 up 107 days, 23:12, 6 users, load average: 1.03, 1.02, 1.12
Tasks: 193 total, 1 running, 192 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.0%us, 2.7%sy, 0.7%ni, 72.1%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1044536k used, 1029420k free, 56988k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 617412k 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>19026</td>
<td>newton</td>
<td>21</td>
<td>0</td>
<td>2496</td>
<td>1100</td>
<td>784</td>
<td>R</td>
<td>2</td>
<td>0.1</td>
<td>0:00.01</td>
<td>top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>18</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532</td>
<td>S</td>
<td>0</td>
<td>0.1</td>
<td>0:08.38</td>
<td>init</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthreadd</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:05.70</td>
<td>migration/0</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:14.79</td>
<td>ksoftirqd/0</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/0</td>
</tr>
<tr>
<td>6</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:04.88</td>
<td>migration/1</td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:04.00</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>RT</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/1</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:01.78</td>
<td>events/0</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>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>15</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:00.02</td>
<td>khelper</td>
</tr>
<tr>
<td>31</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0</td>
<td>0.0</td>
<td>0:01.76</td>
<td>kblockd/0</td>
</tr>
</tbody>
</table>