WaveScript Benchmarks Performance Report

November 12, 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: 3655

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:
## NOSUDD:
## NICE:
## CC:
## 'which icc' =
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2520.000 2516.000 2520.000 2512.000 2516.000 5020.000 5020.000
readfile_bigwins 3768.000 456.000 1140.000 1300.000 4576.000 332.000 788.000
printing_lists 2600.000 900.000 900.000 856.000 868.000 796.000 804.000
conv_SigsegArr 2336.000 396.000 7428.000 856.000 5304.000 36.000 6516.000
fft 140.000 928.000 936.000 936.000 976.000 896.000 880.000
```

Language Shootout:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDD:
## NICE:
## CC:
## 'which icc' =
Benchmark c2
fannkuch2 4532.000
```

Application Benchmarks:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDD:
## NICE:
## CC:
## 'which icc' =
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7548.000 12373.000 4276.000 7592.000 5696.000 9965.000 4000.000
## Running benchmark marmot2.bench for 150 tuples.
run_3phases 9857.000 5912.000 4908.000 5664.000 5248.000 5520.000 4816.000
test_marmot2 2200.000 5296.000 5252.000 4648.000 4704.000 4512.000 4520.000
## Running benchmark marmot3.bench for 14 tuples.
```
# Real or User time for each benchmark/backend
# LD_PRELOAD:
# NOSUDO:
# NICE:
# ‘which icc’ =
Benchmark mltonO3 c2boehm c2 c2def
# Running benchmark bgsub3.bench for 10 tuples.
bgSub3_integer 10277.000 8497.000 9921.000 7824.000

## B Appendix: Additional system information

Top results before running benchmarks:

top - 12:07:45 up 113 days, 21:41, 5 users, load average: 1.00, 1.02, 1.02
Tasks: 190 total, 1 running, 189 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.3%us, 2.6%sy, 0.7%ni, 73.0%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1290704k used, 783252k free, 159796k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 810228k 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 %CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>19652</td>
<td>newton</td>
<td>20</td>
<td>0</td>
<td>2492</td>
<td>1096</td>
<td>784</td>
<td>R 2</td>
<td>0.1</td>
<td>0:00.01</td>
<td>top</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>S 0</td>
<td>0.1</td>
<td>0:08.96</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S 0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</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>S 0</td>
<td>0.0</td>
<td>0:05.86</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 0</td>
<td>0.0</td>
<td>0:14.83</td>
<td>ksoftirqd/0</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>S 0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/0</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>S 0</td>
<td>0.0</td>
<td>0:04.99</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 0</td>
<td>0.0</td>
<td>0:04.01</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S 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 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 0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</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>S 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 0</td>
<td>0.0</td>
<td>0:01.84</td>
<td>kblockd/0</td>
</tr>
</tbody>
</table>

Top results after running benchmarks:

top - 13:09:36 up 113 days, 22:43, 5 users, load average: 1.43, 1.81, 1.67
Tasks: 191 total, 1 running, 189 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.3%us, 2.6%sy, 0.7%ni, 73.0%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1178220k used, 895736k free, 166240k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 702832k 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 %CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>32643</td>
<td>newton</td>
<td>15</td>
<td>0</td>
<td>15836</td>
<td>11m</td>
<td>4200</td>
<td>S 0</td>
<td>0.6</td>
<td>433:53.07</td>
<td>emacs21-x</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>S 0</td>
<td>0.1</td>
<td>0:08.98</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S 0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</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>S 0</td>
<td>0.0</td>
<td>0:05.86</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 0</td>
<td>0.0</td>
<td>0:14.83</td>
<td>ksoftirqd/0</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>S 0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/0</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>S 0</td>
<td>0.0</td>
<td>0:04.99</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 0</td>
<td>0.0</td>
<td>0:04.01</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>34</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S 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 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 0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>events/1</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>S 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 0</td>
<td>0.0</td>
<td>0:01.84</td>
<td>kblockd/0</td>
</tr>
</tbody>
</table>