WaveScript Benchmarks Performance Report

November 5, 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: 3630

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 enusboxes).
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:
## NOSUDD:
## NICE:
## CC:
## ‘which icc’ =
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2552.000 2540.000 2532.000 2524.000 2524.000 4996.000 5032.000
readfile_bigsizes 3696.000 372.000 1100.000 1312.000 3716.000 280.000 832.000
printing_lists 2512.000 936.000 888.000 848.000 908.000 860.000 800.000
conv_SigsegArr 2280.000 396.000 7252.000 820.000 5652.000 64.000 6568.000
fft 128.000 928.000 896.000 816.000 900.000 824.000 976.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDD:
## NICE:
## CC:
## ‘which icc’ =
Benchmark c2
fannkuch2 4312.000

Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDD:
## NICE:
## CC:
## ‘which icc’ =
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7288.000 11737.000 4168.000 7344.000 5576.000 7352.000 3932.000
## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2192.000 5232.000 5268.000 4648.000 4720.000 4532.000 4564.000
## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7776.000 3236.000 3248.000 2564.000 2528.000 3252.000 3248.000
## Running benchmark marmot_all.bench for 20 tuples.
run_3phases 9517.000 5872.000 4860.000 5620.000 5304.000 5564.000 4848.000

## 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 10165.000 8373.000 9849.000 7720.000 7720.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 06:22:32 up 106 days, 15:56, 6 users, load average: 1.01, 1.01, 0.77
Tasks: 196 total, 1 running, 195 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.2%us, 2.7%sy, 0.7%ni, 71.8%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1360760k used, 713196k free, 132256k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 829164k 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>1</td>
<td>root</td>
<td>21</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532</td>
<td>S</td>
<td>0.1</td>
<td>0.0</td>
<td>0:08.27</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>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</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>0.0</td>
<td>0.0</td>
<td>0:05.65</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>0.0</td>
<td>0.0</td>
<td>0:14.78</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>0.0</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>0.0</td>
<td>0.0</td>
<td>0:04.84</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>0.0</td>
<td>0.0</td>
<td>0:03.99</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>0.0</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>0.0</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>0.0</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</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.02</td>
<td>khelper</td>
</tr>
<tr>
<td>31</td>
<td>root</td>
<td>16</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.72</td>
<td>kblockd/0</td>
</tr>
<tr>
<td>32</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>kblockd/1</td>
</tr>
</tbody>
</table>

Top results after running benchmarks:

top - 06:42:47 up 106 days, 16:16, 6 users, load average: 0.93, 0.99, 0.91
Tasks: 196 total, 1 running, 195 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.2%us, 2.7%sy, 0.7%ni, 71.8%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1054960k used, 1018996k free, 56444k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 630572k 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>1</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>2948</td>
<td>1856</td>
<td>532</td>
<td>S</td>
<td>0.1</td>
<td>0.0</td>
<td>0:08.28</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>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</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>0.0</td>
<td>0.0</td>
<td>0:05.65</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>0.0</td>
<td>0.0</td>
<td>0:14.78</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>0.0</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>0.0</td>
<td>0.0</td>
<td>0:04.85</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>0.0</td>
<td>0.0</td>
<td>0:03.99</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>0.0</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>0.0</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>0.0</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</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.02</td>
<td>khelper</td>
</tr>
<tr>
<td>31</td>
<td>root</td>
<td>16</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:01.72</td>
<td>kblockd/0</td>
</tr>
<tr>
<td>32</td>
<td>root</td>
<td>10</td>
<td>-5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.04</td>
<td>kblockd/1</td>
</tr>
</tbody>
</table>