WaveScript Benchmarks Performance Report

September 11, 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: 3566

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 ensembles).

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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mltonO3</th>
<th>c2boehm</th>
<th>c2boehmseglist</th>
<th>c2seglist</th>
<th>c2def</th>
<th>c2defseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td>just_timer</td>
<td>2556.000</td>
<td>2512.000</td>
<td>2524.000</td>
<td>2548.000</td>
<td>2524.000</td>
<td>5024.000</td>
</tr>
<tr>
<td>readfile_bigwins</td>
<td>3836.000</td>
<td>456.000</td>
<td>1132.000</td>
<td>748.000</td>
<td>3860.000</td>
<td>288.000</td>
</tr>
<tr>
<td>printing_lists</td>
<td>2624.000</td>
<td>900.000</td>
<td>888.000</td>
<td>864.000</td>
<td>868.000</td>
<td>816.000</td>
</tr>
<tr>
<td>conv_SigsegArr</td>
<td>2280.000</td>
<td>344.000</td>
<td>7476.000</td>
<td>840.000</td>
<td>5480.000</td>
<td>44.000</td>
</tr>
<tr>
<td>fft</td>
<td>96.000</td>
<td>888.000</td>
<td>920.000</td>
<td>820.000</td>
<td>932.000</td>
<td>860.000</td>
</tr>
</tbody>
</table>

Language Shootout:

## Real or User time for each benchmark/backend

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>c2</th>
</tr>
</thead>
<tbody>
<tr>
<td>fannkuch2</td>
<td>4448.000</td>
</tr>
</tbody>
</table>

Language Shootout:
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 7168.000 11785.000 4172.000 7396.000 5560.000 7216.000 3816.000
## Running marmot2
test_marmot2 2308.000 5272.000 5220.000 4712.000 4704.000 4556.000 4680.000
## Running marmot3
test_heatmap 7756.000 3208.000 3244.000 2540.000 2520.000 3244.000 3252.000
## Running marmot multinode offline
run_3phases 9481.000 5908.000 4824.000 5612.000 5300.000 5496.000 4776.000

B Appendix: Additional system information

Top results before running benchmarks:
top - 06:21:43 up 51 days, 14:55, 3 users, load average: 1.00, 0.99, 0.73
Tasks: 163 total, 1 running, 162 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.7%us, 4.1%sy, 0.8%ni, 66.5%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1561908k used, 512048k free, 149588k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 853968k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:05.67 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.45 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.86 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0 0.0 0:00.35 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.19 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0 0.0 0:00.03 events/0
10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
11 root 18 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:00.59 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1

Top results after running benchmarks:
top - 06:39:59 up 51 days, 15:14, 3 users, load average: 1.10, 1.04, 0.92
Tasks: 162 total, 1 running, 161 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.7%us, 4.1%sy, 0.8%ni, 66.5%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1041564k used, 1032392k free, 6492k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 536400k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:05.67 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.45 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.86 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0 0.0 0:00.35 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.19 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0 0.0 0:00.03 events/0
10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
11 root 18 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:00.59 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1