WaveScript Benchmarks Performance Report

December 3, 2007

Machine information:
Linux faith 2.6.18-4-k7 #1 SMP Wed May 9 23:42:01 UTC 2007 i686 GNU/Linux

WaveScript SVN:
Revision: 2934

WaveScope Engine SVN:
Revision: 1495

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.

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 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.
A Appendix: Raw numbers for above graphs

Microbenchmarks

## User time for each benchmark/backend
Benchmark "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFit DF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"
readfile_bigwins -1 1144 -1 16.0 512.0 -1
dge_stress -1 3496 -1 772.0 52.0 -1
printing_lists -1 6556 -1 1352.0 32.0 -1
convSigsegArr -1 16 -1 844.0 0.0 -1
fft -1 2552 -1 304.0 36.0 -1

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFit DF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"
fannkuch2 -1 16493 -1 736.0 976.0 -1

Application Benchmarks:

## Running orig marmot phase 1
run_first_phase -1 11164 -1 1556.0 404.0 -1
## Running marmot2
test_marmot2 -1 7928 -1 700.0 496.0 -1
## Running marmot3
test_heatmap -1 8116 -1 2768.0 2316.0 -1

B Appendix: Additional system information

Top results before running benchmarks:

top - 06:36:57 up 126 days, 17:09, 13 users, load average: 1.01, 1.05, 1.08
Tasks: 465 total, 1 running, 464 sleeping, 0 stopped, 0 zombie
Cpu(s): 13.2%us, 1.6%sy, 2.6%ni, 81.2%id, 1.6%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1408516k used, 667908k free, 289736k buffers
Swap: 1951856k total, 123940k used, 1827916k free, 611900k 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>28441</td>
<td>newton</td>
<td>18</td>
<td>0</td>
<td>2624</td>
<td>1280</td>
<td>792</td>
<td>R 2</td>
<td>0.1</td>
<td>0:00.02</td>
<td>top</td>
</tr>
<tr>
<td>1</td>
<td>root</td>
<td>15</td>
<td>0</td>
<td>2072</td>
<td>100</td>
<td>72</td>
<td>S 0</td>
<td>0.0</td>
<td>2:00.84</td>
<td>init</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>1</td>
<td>RT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S 0</td>
<td>0.0</td>
<td>0:03.61</td>
<td>migration/0</td>
</tr>
</tbody>
</table>
Top results after running benchmarks:

```
top - 07:05:19 up 126 days, 17:38, 13 users, load average: 1.00, 1.02, 1.00
Tasks: 465 total, 1 running, 464 sleeping, 0 stopped, 0 zombie
Cpu(s): 13.2%us, 1.6%sy, 2.5%ni, 81.2%id, 1.6%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1398816k used, 677608k free, 245848k buffers
Swap: 1951856k total, 123940k used, 1827916k free, 646944k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
32180 newton 19 0 2624 1280 792 R 2 0.1 0:00.02 top
1 root 15 0 2072 100 72 S 0 0.0 2:00.84 init
2 root RT 0 0 0 0 S 0 0.0 0:03.61 migration/0
3 root 34 19 0 0 0 S 0 0.0 0:39.13 ksoftirqd/0
4 root RT 0 0 0 0 S 0 0.0 0:04.57 migration/1
5 root 10 -5 0 0 0 S 0 0.0 0:00.16 events/0
6 root 10 -5 0 0 0 S 0 0.0 0:00.11 events/1
7 root 11 -5 0 0 0 S 0 0.0 0:00.00 khelper
8 root 19 -5 0 0 0 S 0 0.0 0:00.01 kthread
9 root 10 -5 0 0 0 S 0 0.0 0:00.72 kblockd/0
10 root 16 -5 0 0 0 S 0 0.0 0:01.19 kblockd/1
11 root 11 -5 0 0 0 S 0 0.0 0:00.00 kacpid
12 root 10 -5 0 0 0 S 0 0.0 0:00.00 kacpid
13 root 10 -5 0 0 0 S 0 0.0 0:00.00 kacpid
14 root 10 -5 0 0 0 S 0 0.0 0:00.00 kacpid
15 root 15 -5 0 0 0 S 0 0.0 0:00.00 kacpid
```