WaveScript Benchmarks Performance Report

January 4, 2008

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: 2983

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 enboxes).

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" "CoreFitDF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"
readfile_bigwins 2432 1152 276.0 4.0 4.0 8.0 504.0 432.0
printing_lists 11136 1724 688.0 664.0 656.0 612.0 1060.0 1068.0
conv_SigsegArr 27150 20 996.0 972.0 868.0 852.0 8.0 4.0
fft 4528 2688 372.0 348.0 340.0 320.0 40.0 36.0

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" "CoreFitDF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"
fannkuch2 15692 12061 736.0 732.0 732.0 728.0 976.0 712.0

Application Benchmarks:

Benchmark "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFitDF -j 1" "CoreFitDF 1Thread -j 1"
run_first_phase 29354 11017 4108.0 1824.0 1812.0 1524.0 412.0 344.0
test_marmot2 13041 7620 740.0 704.0 764.0 804.0 484.0 492.0
test_heatmap 12796 7740 3268.0 3252.0 3144.0 2864.0 2320.0 2372.0

B Appendix: Additional system information

Top results before running benchmarks:
top - 11:31:56 up 158 days, 22:05, 5 users, load average: 1.08, 1.02, 0.92
Tasks: 161 total, 1 running, 153 sleeping, 6 stopped, 1 zombie
Cpu(s): 14.1%us, 1.8%sy, 2.0%ni, 80.7%id, 1.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1358868k used, 717556k free, 482100k buffers
Swap: 1951856k total, 1358868k used, 717556k free, 482100k buffers

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 20 0 2092 88 60 S 0 0.0 2:07.92 init
2 root RT 0 0 0 0 S 0 0.0 0:05.10 migration/0
3 root 34 19 0 0 0 S 0 0.0 0:40.10 ksoftirqd/0
4 root RT 0 0 0 0 S 0 0.0 0:06.03 migration/1
Top results after running benchmarks:

```
top - 12:02:34 up 158 days, 22:35, 5 users, load average: 1.00, 1.00, 1.00
Tasks: 162 total, 1 running, 154 sleeping, 6 stopped, 1 zombie
Cpu(s): 14.1%us, 1.8%sy, 2.0%ni, 80.7%id, 1.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1398680k used, 677744k free, 442984k buffers
Swap: 1951856k total, 477744k free, 1904080k used, 371436k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 15 0 2092 88 60 S 0 0.0 2:07.92 init
2 root RT 0 0 0 0 S 0 0.0 0:05.11 migration/0
3 root 34 19 0 0 0 S 0 0.0 0:40.10 ksoftirqd/0
4 root RT 0 0 0 0 S 0 0.0 0:06.03 migration/1
5 root 34 19 0 0 0 S 0 0.0 1:59.59 ksoftirqd/1
6 root 10 -5 0 0 0 S 0 0.0 0:00.16 events/0
7 root 10 -5 0 0 0 S 0 0.0 0:00.12 events/1
8 root 14 -5 0 0 0 S 0 0.0 0:00.00 khelper
9 root 10 -5 0 0 0 S 0 0.0 0:00.02 kthread
13 root 10 -5 0 0 0 S 0 0.0 0:00.96 kblockd/0
14 root 14 -5 0 0 0 S 0 0.0 0:01.46 kblockd/1
15 root 15 -5 0 0 0 S 0 0.0 0:00.00 kacpid
82 root 12 -5 0 0 0 S 0 0.0 0:00.00 kseriod
```