WaveScript Benchmarks Performance Report

January 3, 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: 2982

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 2452 1204 312.0 4.0 8.0 4.0 500.0 412.0
printing_lists 11164 1684 700.0 672.0 652.0 604.0 1120.0 1112.0
cnv_SigsegArr 26029 56 1000.0 976.0 864.0 860.0 0.0 4.0
fft 4532 2524 404.0 364.0 364.0 336.0 32.0 32.0

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" "CoreFit DF -j 1" "CoreFitDF 1Thread -j 1" "MLton -O2" "MLton -O3"
fannkuch2 15857 12249 736.0 732.0 732.0 724.0 972.0 712.0

Application Benchmarks:

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"
run_first_phase 29326 11105 4288.0 1888.0 1900.0 1592.0 412.0 356.0
test_marmot2 13689 7420 740.0 712.0 720.0 800.0 496.0 492.0
test_heatmap 12621 7680 3224.0 3196.0 3144.0 2880.0 2328.0 2372.0

B Appendix: Additional system information

Top results before running benchmarks:
top - 06:43:45 up 157 days, 17:16, 6 users, load average: 1.01, 1.01, 1.03
Tasks: 164 total, 1 running, 156 sleeping, 6 stopped, 1 zombie
Cpu(s): 14.2%us, 1.8%sy, 2.0%ni, 80.6%id, 1.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1405488k used, 670936k free, 96612k buffers
Swap: 1951856k total, 47296k used, 1904560k free, 1003756k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
17463 newton 17 0 2516 1112 800 R 2 0.1 0:00.01 top
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 0:05.10 migration/0
3 root 36 19 0 0 0 S 0 0.0 0:40.10 ksoftirqd/0
Top results *after* running benchmarks:

```
top - 07:14:19 up 157 days, 17:47, 6 users, load average: 1.07, 1.02, 1.00
Tasks: 164 total, 1 running, 156 sleeping, 6 stopped, 1 zombie
Cpu(s): 14.2%us, 1.8%sy, 2.0%ni, 80.6%id, 1.5%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1397000k used, 679424k free, 101944k buffers
Swap: 1951856k total, 479424k free, 101944k buffers
```

```
  PID USER   PR  NI  VIRT  RES   SHR  S %CPU %MEM   TIME+ COMMAND
  1 root    20  0   0    88   60  S   0.0  0.0   2:07.92  init
  2 root    19  0   0    0    0  S   0.0  0.0   0:05.10  migration/0
  3 root    19  0   0    0    0  S   0.0  0.0   0:40.10  ksoftirqd/0
  4 root    19  0   0    0    0  S   0.0  0.0   0:06.02  migration/1
  5 root    19  0   0    0    0  S   0.0  0.0   1:59.59  ksoftirqd/1
  6 root    19  0   0    0    0  S   0.0  0.0   0:00.16  events/0
  7 root    19  0   0    0    0  S   0.0  0.0   0:00.12  events/1
  8 root    19  0   0    0    0  S   0.0  0.0   0:00.00  khelper
  9 root    19  0   0    0    0  S   0.0  0.0   0:00.02  kthread
 10 root    19  0   0    0    0  S   0.0  0.0   0:00.96  kblockd/0
 11 root    19  0   0    0    0  S   0.0  0.0   0:01.46  kblockd/1
 12 root    19  0   0    0    0  S   0.0  0.0   0:00.00  kacpid
 13 root    19  0   0    0    0  S   0.0  0.0   0:00.00  kseriod
```