WaveScript Benchmarks Performance Report

December 6, 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: 2946

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 backends. 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 1148 -1 4.0 484.0 -1
dge_stress -1 3546 -1 772.0 52.0 -1
printing_lists -1 6592 -1 1432.0 1724.0 -1
conv_SigsegArr -1 16 -1 860.0 4.0 -1
fft -1 2588 -1 316.0 32.0 -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 17069 -1 736.0 976.0 -1

Application Benchmarks:

## Running orig marmot phase 1
run_first_phase -1 11124 -1 1708.0 396.0 -1
## Running marmot2
test_marmot2 -1 7992 -1 808.0 500.0 -1
## Running marmot3
test_heatmap -1 8388 -1 2796.0 2380.0 -1

B Appendix: Additional system information

Top results before running benchmarks:
top - 06:41:10 up 129 days, 17:14, 9 users, load average: 2.01, 2.07, 2.24
Tasks: 183 total, 2 running, 181 sleeping, 0 stopped, 0 zombie
Cpu(s): 13.5%us, 1.5%sy, 2.4%ni, 80.9%id, 1.6%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424kb total, 1406576kb used, 669848kb free, 83752kb buffers
Swap: 1951856kb total, 315840kb used, 1636016kb free, 1057528kb cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
24586 mkinsy 25 0 25952 23m 1532 R 99 1.2 234:39.72 perl
5685 newton 16 0 2516 1120 800 R 2 0.1 0:00:01 top
1 root 15 0 2092 128 104 S 0 0.0 2:00.87 init
Top results after running benchmarks:

```
Top results after running benchmarks:

Tasks: 183 total, 2 running, 181 sleeping, 0 stopped, 0 zombie
Cpu(s): 13.5%us, 1.5%sy, 2.4%ni, 80.9%id, 1.6%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1398564k used, 677860k free, 90744k buffers
Swap: 1951856k total, 315816k used, 1636040k free, 1036168k cached

2 root   RT  0   0   0   0 S   0   0.0 0:03.74 migration/0
3 root   34  19   0   0   0 S   0   0.0 0:39.20 ksoftirqd/0
4 root   RT  0   0   0   0 S   0   0.0 0:04.68 migration/1
5 root   39  19   0   0   0 S   0   0.0 1:56.16 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   13 -5   0   0   0 S   0   0.0 0:00.01 kthread
13 root  10 -5   0   0   0 S   0   0.0 0:00.76 kblockd/0
14 root  10 -5   0   0   0 S   0   0.0 0:01.28 kblockd/1
```

Top results after running benchmarks:

top - 07:10:06 up 129 days, 17:43, 9 users, load average: 2.02, 2.03, 2.06
Tasks: 183 total, 2 running, 181 sleeping, 0 stopped, 0 zombie
Cpu(s): 13.5%us, 1.5%sy, 2.4%ni, 80.9%id, 1.6%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1398564k used, 677860k free, 90744k buffers
Swap: 1951856k total, 315816k used, 1636040k free, 1036168k cached

```
24586  mkinsy  25  0  26216  23m  1532 R   99 1.2 263:33.84 perl
1 root   15   0  2092  128  104 S   0   0.0 2:00.87 init
2 root   RT  0   0   0   0 S   0   0.0 0:03.74 migration/0
3 root   34  19   0   0   0 S   0   0.0 0:39.20 ksoftirqd/0
4 root   RT  0   0   0   0 S   0   0.0 0:04.68 migration/1
5 root   34  19   0   0   0 S   0   0.0 1:56.16 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   13 -5   0   0   0 S   0   0.0 0:00.01 kthread
13 root  10 -5   0   0   0 S   0   0.0 0:00.76 kblockd/0
14 root  10 -5   0   0   0 S   0   0.0 0:01.28 kblockd/1
15 root  15 -5   0   0   0 S   0   0.0 0:00.00 kacpid
```