WaveScript Benchmarks Performance Report

August 24, 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: 3528

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.

![fannkuch2](image)

## 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 mltonO3 c2boehm c2boehmselist c2 c2seglst c2def c2defseglst
just_timer 2556.000 2556.000 2560.000 2524.000 2560.000 5520.000 5048.000
readfile_bigwins 3692.000 496.000 1120.000 1372.000 3936.000 312.000 1004.000
printing_lists 2584.000 920.000 932.000 840.000 908.000 812.000 836.000
conv_SigsegArr 2316.000 320.000 7388.000 840.000 5292.000 60.000 6428.000
fft 116.000 1016.000 924.000 892.000 1004.000 888.000 900.000

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4464.000

Application Benchmarks:

Benchmark mltonO3 c2boehm c2boehmselist c2 c2seglst c2def c2defseglst
## Running orig marmot phase 1
## Running marmot2

test_marmot2 2232.000 5224.000 5248.000 4656.000 4732.000 4576.000 4568.000

## Running marmot3
test_heatmap 8181.000 3288.000 3256.000 2540.000 2556.000 3272.000 3232.000

## Running marmot multinode offline
run_3phases 9697.000 6000.000 4824.000 5660.000 5312.000 5604.000 4888.000

---

### Appendix: Additional system information

Top results before running benchmarks:

```
top - 10:10:45 up 33 days, 18:44,  5 users, load average: 1.01, 1.22, 1.01
Tasks: 156 total,  1 running, 155 sleeping,  0 stopped,  0 zombie
Cpu(s):  24.8%us,  4.2%sy,   1.2%ni, 68.9%id,  0.1%wa,  0.4%hi,  0.4%si,  0.0%st
Mem: 2073956k total, 1367444k used,  706512k free,  113800k buffers
Swap: 14996668k total,  34740k used, 14961928k free,  906760k cached
```

```
PID USER   PR NI  VIRT  RES   SHR S %CPU %MEM    TIME+ COMMAND
 1 root    20  0  2948  1856  532 S   0.1  0:04.80    init
 2 root    11 -5  0   0   0  S   0.0  0:00.00  kthreadd
 3 root RT -5  0   0   0  S   0.0  0:00.30  migration/0
 4 root  34 19  0   0   0  S   0.0  0:00.64  ksoftirqd/0
 5 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/0
 6 root RT -5  0   0   0  S   0.0  0:00.25  migration/1
 7 root  34 19  0   0   0  S   0.0  0:01.09  ksoftirqd/1
 8 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/1
 9 root  10 -5  0   0   0  S   0.0  0:00.03  events/0
10 root  10 -5  0   0   0  S   0.0  0:00.04  events/1
11 root  10 -5  0   0   0  S   0.0  0:00.02  khelper
31 root  10 -5  0   0   0  S   0.0  0:00.40  kblockd/0
32 root  10 -5  0   0   0  S   0.0  0:00.00  kblockd/1
```

Top results after running benchmarks:

```
top - 10:28:51 up 33 days, 19:02,  5 users, load average: 1.06, 1.06, 1.00
Tasks: 156 total,  1 running, 155 sleeping,  0 stopped,  0 zombie
Cpu(s):  24.8%us,  4.2%sy,   1.2%ni, 68.9%id,  0.1%wa,  0.4%hi,  0.4%si,  0.0%st
Mem: 2073956k total, 1043312k used,  706512k free,  116912k buffers
Swap: 14996668k total,  34740k used, 14961928k free,  609772k cached
```

```
PID USER   PR NI  VIRT  RES   SHR S %CPU %MEM    TIME+ COMMAND
 1 root    15  0  2948  1856  532 S   0.1  0:04.80    init
 2 root    11 -5  0   0   0  S   0.0  0:00.00  kthreadd
 3 root RT -5  0   0   0  S   0.0  0:00.30  migration/0
 4 root  34 19  0   0   0  S   0.0  0:00.65  ksoftirqd/0
 5 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/0
 6 root RT -5  0   0   0  S   0.0  0:00.25  migration/1
 7 root  34 19  0   0   0  S   0.0  0:01.09  ksoftirqd/1
 8 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/1
 9 root  10 -5  0   0   0  S   0.0  0:00.03  events/0
10 root  10 -5  0   0   0  S   0.0  0:00.04  events/1
11 root  10 -5  0   0   0  S   0.0  0:00.02  khelper
31 root  10 -5  0   0   0  S   0.0  0:00.40  kblockd/0
32 root  10 -5  0   0   0  S   0.0  0:00.00  kblockd/1
```