WaveScript Benchmarks Performance Report

September 5, 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: 3559

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.

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

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mltonO3 c2boehm c2boehmseglst c2 c2seglist c2def c2defseglst
just_timer 2556.000 2524.000 2540.000 2512.000 2516.000 5036.000 5036.000
readfile_bigwins 3764.000 508.000 1108.000 740.000 3956.000 296.000 1064.000
printing_lists 2596.000 900.000 908.000 876.000 840.000 848.000 808.000
conv_SigsegArr 2268.000 380.000 7304.000 820.000 5464.000 52.000 7060.000
fft 124.000 916.000 956.000 976.000 988.000 988.000 900.000 888.000
```

Language Shootout:

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4540.000
```
Application Benchmarks:

## Real or User time for each benchmark/backend

## LD_PRELOAD:
## NOSUDO:
## NICE:

Benchmark mlt0nO3 c2boehm c2boehmsseglist c2 c2seglist c2def c2defseglist

## Running orig marmot phase 1
run_first_phase 7128.000 11901.000 4196.000 7420.000 5640.000 7212.000 3832.000

## Running marmot2
test_marmot2 2364.000 5260.000 5240.000 4704.000 4676.000 4564.000 4580.000

## Running marmot3
test_heatmap 7748.000 3224.000 3232.000 2544.000 2548.000 3208.000 3200.000

## Running marmot multinode offline
run_3phases 9473.000 5916.000 4864.000 5608.000 5324.000 5496.000 4836.000

B Appendix: Additional system information

Top results before running benchmarks:
top - 11:15:44 up 45 days, 19:49, 4 users, load average: 1.20, 1.13, 1.09
Tasks: 159 total, 1 running, 158 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.3%us, 4.4%sy, 0.9%ni, 66.4%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1565760k used, 508196k free, 176904k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 1002568k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
27338 newton 21 0 2368 1084 784 R 2 0.1 0:00.01 top
30134 newton 15 0 34008 25a 4048 S 2 1.3 293:04.82 unison-2.18
1 root 18 0 2948 1856 532 S 0 0.1 0:06.42 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
3 root RT -5 0 0 0 S 0 0.0 0:00.41 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.84 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0 0.0 0:00.33 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.18 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0 0.0 0:00.03 events/0
10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
11 root 10 -5 0 0 0 S 0 0.0 0:00.02 khelper

Top results after running benchmarks:
top - 11:34:14 up 45 days, 20:08, 4 users, load average: 1.12, 1.14, 1.09
Tasks: 165 total, 1 running, 164 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.3%us, 4.4%sy, 0.9%ni, 66.4%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1088404k used, 985552k free, 96036k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 621528k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
30134 newton 15 0 34008 25m 4048 S 6 1.3 294:30.23 unison-2.18
1 root 15 0 2948 1856 532 S 0 0.1 0:05.43 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
3 root RT -5 0 0 0 S 0 0.0 0:00.42 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.84 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0 0.0 0:00.33 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.18 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0 0.0 0:00.03 events/0
10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
11 root 10 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:00.54 kblockd/0