WaveScript Benchmarks Performance Report

September 9, 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: 3562

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 mlton03 c2boehm c2boehmsiglist c2 c2siglist c2def c2defsiglist
just_timer 2564.000 2532.000 2552.000 2516.000 2536.000 5020.000 5024.000
readfile_bigwins 3808.000 432.000 1088.000 740.000 3880.000 248.000 1004.000
printing_lists 2720.000 876.000 920.000 920.000 852.000 828.000 840.000
conv_SigSegArr 2324.000 456.000 7320.000 852.000 5480.000 40.000 6644.000
fft 140.000 880.000 920.000 764.000 880.000 880.000 868.000 856.000
```

Language Shootout:

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4448.000
```
Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
run_first_phase 7132.000 11753.000 4172.000 7316.000 5592.000 7236.000 3804.000
## Running marmot2
test_marmot2 2344.000 5224.000 5220.000 4720.000 4700.000 4572.000 4572.000
## Running marmot3
test_heatmap 7764.000 3232.000 3224.000 2560.000 2532.000 3240.000 3264.000
## Running marmot multinode offline
run_3phases 9497.000 5892.000 4856.000 5600.000 5300.000 5528.000 4764.000

B Appendix: Additional system information

Top results before running benchmarks:

```
top - 12:55:35 up 49 days, 21:29, 3 users, load average: 0.91, 0.98, 0.84
Tasks: 154 total, 2 running, 152 sleeping, 0 stopped, 0 zombie
Cpu(s): 28.2%us, 4.2%sy, 0.8%ni, 65.8%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1184228k used, 889728k free, 113532k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 686284k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
5824 avahi 15 0 2852 1472 1204 S 2 0.1 0:27.31 avahi-daemon
32643 newton 15 0 15836 11a 4200 S 2 0.6 64:26.26 emacs21-x
1 root 18 0 2948 1856 532 S 0 0.1 0:05.53 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.44 migration/0
4 root 34 19 0 0 0 R 0 0.0 0:00.86 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 0:00.34 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.19 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 19 -5 0 0 0 S 0 0.0 0:00.02 khelper
```

Top results after running benchmarks:

```
top - 13:13:50 up 49 days, 21:47, 3 users, load average: 1.08, 1.28, 1.18
Tasks: 155 total, 1 running, 154 sleeping, 0 stopped, 0 zombie
Cpu(s): 28.2%us, 4.2%sy, 0.8%ni, 65.8%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1029032k used, 1044924k free, 86884k buffers
Swap: 14996668k total, 14961924k free, 570904k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:05.55 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.44 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.86 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 0:00.34 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.19 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 19 -5 0 0 0 S 0 0.0 0:00.02 khelper
```

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
31 root 10 -5 0 0 0 S 0 0.0 0:00.58 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1