WaveScript Benchmarks Performance Report

October 23, 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: 3604

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 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

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2576.000 2508.000 2508.000 2528.000 2520.000 5028.000 5032.000
readfile_bigwins 3820.000 432.000 1000.000 1296.000 3784.000 268.000 1048.000
printing_lists 2752.000 928.000 944.000 876.000 936.000 828.000 816.000
conv_SigsegArr 2300.000 456.000 7252.000 844.000 5788.000 44.000 6416.000
fft 128.000 892.000 932.000 808.000 962.000 880.000 964.000
```

Language Shootout:

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4332.000
```
Application Benchmarks:
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mltonO3 c2boehm c2boehmseglst c2 c2seglst c2def c2defseglst
## Running orig marmot phase 1
run_first_phase 7160.000 11901.000 4120.000 7312.000 5528.000 7252.000 3736.000
## Running marmot2
test_marmot2 2364.000 5268.000 5272.000 4656.000 4676.000 4576.000 4572.000
## Running marmot3
test_heatmap 7740.000 3208.000 3252.000 2536.000 2576.000 3244.000 3252.000
## Running marmot multinode offline
run_3phases 9501.000 5868.000 4856.000 5584.000 5284.000 5480.000 4804.000

B Appendix: Additional system information

Top results before running benchmarks:
```
top - 06:22:13 up 93 days, 14:56, 4 users, load average: 1.00, 1.02, 0.77
Tasks: 187 total, 1 running, 186 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.9%us, 3.0%sy, 0.8%ni, 68.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1291964k used, 781992k free, 107288k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 793172k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:07.49 init
2 root 13 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:04.89 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.70 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 0:04.08 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:03.96 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 0:01.78 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 0:01.50 kblockd/0
32 root 14 -5 0 0 0 S 0 0.0 0:00.03 kblockd/1
```

Top results after running benchmarks:
```
top - 06:41:09 up 93 days, 15:15, 4 users, load average: 1.01, 1.01, 0.91
Tasks: 187 total, 1 running, 186 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.9%us, 3.0%sy, 0.8%ni, 68.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1041216k used, 1032740k free, 63172k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 604636k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:07.52 init
2 root 13 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:04.89 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.70 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 0:04.09 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:03.96 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 0:01.78 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 0:01.50 kblockd/0
32 root 14 -5 0 0 0 S 0 0.0 0:00.03 kblockd/1
```