WaveScript Benchmarks Performance Report

August 13, 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: 3473

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

## User time for each benchmark/backend
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 148.000 148.000 124.000 152.000 152.000 252.000 256.000
readfile_bigwins 2252.000 236.000 264.000 12.000 16.000 24.000 12.000
printing_lists 1304.000 440.000 468.000 436.000 420.000 404.000 424.000
conv_SigsegArr 2320.000 384.000 7252.000 804.000 6432.000 32.000 6400.000
fft 68.000 484.000 456.000 440.000 476.000 436.000 476.000

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4516.000

Application Benchmarks:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
B Appendix: Additional system information

Top results before running benchmarks:

```
top - 06:14:01 up 22 days, 14:48, 5 users, load average: 0.92, 0.93, 0.59
Tasks: 161 total, 1 running, 160 sleeping, 0 stopped, 0 zombie
Cpu(s): 34.7%us, 4.5%sy, 1.6%ni, 58.2%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1556196k used, 515760k free, 200752k buffers
Swap: 1499668k total, 34740k used, 14961928k free, 862688k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:04.22 init
2 root 10 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.20 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.50 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 0:00.14 migration/1
7 root 39 19 0 0 0 S 0 0.0 0:00.92 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 0:00.00 khelper
31 root 15 -5 0 0 0 S 0 0.0 0:00.23 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1
```

Top results after running benchmarks:

```
top - 06:30:36 up 22 days, 15:04, 5 users, load average: 1.16, 1.08, 0.87
Tasks: 161 total, 1 running, 160 sleeping, 0 stopped, 0 zombie
Cpu(s): 34.7%us, 4.5%sy, 1.6%ni, 58.1%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1119904k used, 954052k free, 199916k buffers
Swap: 1499668k total, 34740k used, 14961928k free, 476388k cached
```

```
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 16 0 2948 1856 532 S 0 0.1 0:04.22 init
2 root 10 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.20 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.50 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 0:00.14 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:00.92 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 0:00.00 khelper
31 root 15 -5 0 0 0 S 0 0.0 0:00.23 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1
```

6