WaveScript Benchmarks Performance Report

August 11, 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: 3465

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

```plaintext
## User time for each benchmark/backend
Benchmark mltonO3 c2boehm c2boehmsglst c2 c2seglist c2def c2defseglist
just_timer 164.000 176.000 144.000 148.000 176.000 252.000 280.000
readfile_bigwins 2252.000 136.000 172.000 8.000 0 8.000 8.000
printing_lists 1272.000 624.000 616.000 544.000 552.000 536.000 500.000
conv_SigsegArr 2316.000 184.000 7132.000 812.000 5336.000 48.000 6616.000
fft 84.000 432.000 472.000 456.000 492.000 452.000 460.000
```

Language Shootout:

```plaintext
## User time for each language-shootout benchmark/backend
Benchmark c2
fannkuch2 4396.000
```

Application Benchmarks:

```plaintext
Benchmark mltonO3 c2boehm c2boehmsglst c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
```
## Running marmot2

```
run_first_phase 1484.000 2336.000 932.000 1512.000 1108.000 1452.000 792.000
```

## Running marmot3

```
test_marmot2 2348.000 5240.000 5296.000 4656.000 4664.000 4568.000 4560.000
```

## Running marmot heatmap

```
test_heatmap 7844.000 4308.000 4348.000 3960.000 4016.000 4224.000 4180.000
```

## Running marmot multinode offline

```
run_3phases 9293.000 6912.000 5880.000 6216.000 5920.000 6484.000 5816.000
```

B Appendix: Additional system information

### Top results before running benchmarks:

```
top - 14:09:58 up 20 days, 22:44, 6 users, load average: 1.02, 1.01, 0.62
Tasks: 153 total, 1 running, 152 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.3%us, 4.7%sy, 1.9%ni, 56.1%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1550992k used, 522964k free, 51404k buffers

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 15  0 2948 1856  532 S  0  0.1 0:04.06 init
2 root 14 -5  0  0  0 S  0  0.0 0:00.00 kthreadd
3 root RT -5  0  0  0 S  0  0.0 0:00.18 migration/0
4 root 34 19  0  0  0 S  0  0.0 0:00.45 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 0:00.12 migration/1
7 root 37 19  0  0  0 S  0  0.0 0:00.84 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 0:00.03 events/1
11 root 10 -5  0  0  0 S  0  0.0 0:00.00 khelper
31 root 10 -5  0  0  0 S  0  0.0 0:00.22 kblockd/0
32 root 10 -5  0  0  0 S  0  0.0 0:00.00 kblockd/1
```

### Top results after running benchmarks:

```
top - 14:26:36 up 20 days, 23:00, 6 users, load average: 1.00, 1.01, 0.86
Tasks: 153 total, 1 running, 152 sleeping, 0 stopped, 0 zombie
Cpu(s): 36.3%us, 4.7%sy, 1.9%ni, 56.1%id, 0.1%wa, 0.4%hi, 0.5%si, 0.0%st
Mem: 2073956k total, 1550992k used, 522964k free, 51404k buffers
Swap: 1499868k total, 34744k used, 1496424k free, 101172k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18  0 2948 1856  532 S  0  0.1 0:04.06 init
2 root 14 -5  0  0  0 S  0  0.0 0:00.00 kthreadd
3 root RT -5  0  0  0 S  0  0.0 0:00.18 migration/0
4 root 34 19  0  0  0 S  0  0.0 0:00.45 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 0:00.12 migration/1
7 root 34 19  0  0  0 S  0  0.0 0:00.84 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 0:00.03 events/1
11 root 10 -5  0  0  0 S  0  0.0 0:00.00 khelper
31 root 10 -5  0  0  0 S  0  0.0 0:00.22 kblockd/0
32 root 10 -5  0  0  0 S  0  0.0 0:00.00 kblockd/1
```