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: 3605

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.

![Diagram](chart.png)

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 mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer  2532.000  2516.000  2516.000  2508.000  2524.000  5028.000  5048.000
readfile_bigwins  3732.000  424.000  1088.000  1284.000  3744.000  252.000  1004.000
printing_lists  2940.000  932.000  916.000  856.000  868.000  824.000  828.000
conv_SigsegArr  2412.000  356.000  7292.000  836.000  5728.000  60.000  6384.000
fft  144.000  944.000  1008.000  908.000  960.000  888.000  868.000
```

Language Shootout:

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2  4320.000
```
Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
run_first_phase 7128.000 11905.000 4064.000 7324.000 5516.000 7308.000 3808.000
## Running marmot2
test_marmot2 2340.000 5248.000 5252.000 4632.000 4720.000 4556.000 4588.000
## Running marmot3
test_heatmap 7760.000 3236.000 3212.000 2504.000 2500.000 3240.000 3208.000
## Running marmot multinode offline
run_3phases 9437.000 5864.000 4864.000 5568.000 5252.000 5512.000 4952.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 13:55:18 up 93 days, 22:29, 5 users, load average: 1.00, 1.01, 0.73
Tasks: 187 total, 1 running, 186 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.8%us, 3.0%sy, 0.8%ni, 68.8%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1561488k used, 512468k free, 113024k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 1056016k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:07.54 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 0:04.10 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 0:01.53 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.03 kblockd/1

Top results after running benchmarks:

top - 14:14:15 up 93 days, 22:48, 5 users, load average: 1.00, 1.02, 0.91
Tasks: 187 total, 1 running, 186 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.8%us, 3.0%sy, 0.8%ni, 68.8%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1054220k used, 1019736k free, 80808k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 590500k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:07.54 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 0:04.10 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 0:01.53 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.03 kblockd/1