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: 3607

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

## Real or User time for each benchmark/backend

### LD_PRELOAD:
### NOSUDO:
### NICE:

Benchmark | mltonO3 | c2boehm | c2boehmseglist | c2 | c2seglist | c2def | c2defseglist
--- | --- | --- | --- | --- | --- | --- | ---
just_timer | 2580.000 | 2552.000 | 2528.000 | 2504.000 | 2524.000 | 5040.000 | 5044.000
readfile_bigwins | 3648.000 | 476.000 | 1124.000 | 1316.000 | 3760.000 | 276.000 | 1160.000
printing_lists | 2760.000 | 856.000 | 920.000 | 900.000 | 852.000 | 856.000 | 832.000
conv_SigsegArr | 2320.000 | 372.000 | 7272.000 | 804.000 | 5636.000 | 32.000 | 6360.000
fft | 148.000 | 916.000 | 880.000 | 976.000 | 980.000 | 872.000 | 944.000

Language Shootout:

## Real or User time for each benchmark/backend

### LD_PRELOAD:
### NOSUDO:
### NICE:

Benchmark | c2
--- | ---
fannkuch2 | 4328.000
Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:

Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist

## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7300.000 11905.000 4156.000 7376.000 5512.000 7252.000 3732.000

## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2180.000 5228.000 5228.000 4648.000 4720.000 4588.000 4580.000

## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7768.000 3228.000 3248.000 2544.000 2528.000 3212.000 3212.000

## Running benchmark marmot_all.bench for 20 tuples.
run_3phases 9453.000 5900.000 4836.000 5604.000 5284.000 5544.000 4788.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 20:37:12 up 94 days, 5:11, 6 users, load average: 0.85, 0.97, 0.75
Tasks: 190 total, 1 running, 189 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, 1628480k used, 445476k free, 30860k buffers
Swap: 14996668k total, 34760k used, 14961908k free, 1310312k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:07.58 init
2 root 13 0 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:05.19 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.72 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 0:04.35 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:03.98 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 0 0 0 0 S 0 0.0 0:01.78 events/0
10 root 10 0 0 0 0 S 0 0.0 0:00.04 events/1
11 root 10 0 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 0 0 0 0 S 0 0.0 0:01.56 kblockd/0
32 root 10 0 0 0 0 S 0 0.0 0:00.04 kblockd/1

Top results after running benchmarks:

top - 20:56:13 up 94 days, 5:30, 6 users, load average: 1.00, 1.00, 0.91
Tasks: 190 total, 1 running, 189 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, 1397480k used, 676476k free, 21760k buffers
Swap: 14996668k total, 34760k used, 14961908k free, 1091216k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
14031 newton 21 0 2492 1096 794 R 2 0.1 0:00.01 top
1 root 18 0 2948 1856 532 S 0 0.1 0:07.59 init
2 root 13 0 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:05.19 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.72 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 0:04.35 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:03.98 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 0 0 0 0 S 0 0.0 0:01.78 events/0
10 root 10 0 0 0 0 S 0 0.0 0:00.04 events/1
11 root 10 0 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 0 0 0 0 S 0 0.0 0:01.56 kblockd/0
32 root 10 0 0 0 0 S 0 0.0 0:00.04 kblockd/1