WaveScript Benchmarks Performance Report

October 2, 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: 3581

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

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2504.000 2492.000 2516.000 2512.000 2516.000 5024.000 5064.000
readfile_bigwins 3804.000 484.000 1164.000 776.000 3868.000 268.000 1056.000
printing_lists 2560.000 884.000 884.000 856.000 864.000 800.000 808.000
conv_SigsegArr 2268.000 360.000 7272.000 796.000 5524.000 40.000 6608.000
fft 116.000 960.000 884.000 924.000 980.000 868.000 872.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4460.000
Application Benchmarks:

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

Benchmark mltonO3 c2boehm c2boehmsgclist c2 c2seglist c2def c2defseglist
## Running orig marmot phase 1
run_first_phase 7204.000 12265.000 4064.000 7308.000 5540.000 7172.000 3864.000
## Running marmot2
test_marmot2 2328.000 5240.000 5236.000 4700.000 4724.000 4536.000 4548.000
## Running marmot3
test_heatmap 7780.000 3208.000 3252.000 2540.000 2536.000 3248.000 3224.000
## Running marmot multinode offline
run_3phases 9485.000 5872.000 4848.000 5620.000 5276.000 5496.000 4800.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 06:21:41 up 72 days, 14:55, 2 users, load average: 1.00, 1.02, 0.75
Tasks: 164 total, 1 running, 163 sleeping, 0 stopped, 0 zombie
Cpu(s): 32.1%us, 3.7%sy, 1.1%ni, 62.5%id, 0.2%wa, 0.2%hi, 0.3%si, 0.0%st
Mem: 2073956k total, 1573808k used, 500148k free, 134560k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 954712k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
20524 newton 20 0 2364 1080 784 R 2 0.1 0:00.01 top
1 root 18 0 2948 1856 532 S 0 0.1 0:07.00 init
2 root 11 -5 0 0 0 S 0.0 0.00.00 kthread
3 root RT -5 0 0 0 S 0.0 0.04.80 migration/0
4 root 34 19 0 0 0 S 0.0 0.14.61 ksoftirqd/0
5 root RT -5 0 0 0 S 0.0 0.00.00 watchdog/0
6 root RT -5 0 0 0 S 0.0 0.03.94 migration/1
7 root 34 19 0 0 0 S 0.0 0.03.86 ksoftirqd/1
8 root RT -5 0 0 0 S 0.0 0.00.00 watchdog/1
9 root 10 -5 0 0 0 S 0.0 0.00.03 events/0
10 root 10 -5 0 0 0 S 0.0 0.04.00 events/1
11 root 10 -5 0 0 0 S 0.0 0.00.02 khelper
31 root 12 -5 0 0 0 S 0.0 0.01.16 kblockd/0

Top results after running benchmarks:

top - 06:39:57 up 72 days, 15:14, 2 users, load average: 1.00, 1.00, 0.91
Tasks: 164 total, 1 running, 163 sleeping, 0 stopped, 0 zombie
Cpu(s): 32.1%us, 3.7%sy, 1.1%ni, 62.5%id, 0.2%wa, 0.2%hi, 0.3%si, 0.0%st
Mem: 2073956k total, 1049560k used, 1024396k free, 100904k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 487808k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0.1 0:07.00 init
2 root 11 -5 0 0 0 S 0.0 0.00.00 kthread
3 root RT -5 0 0 0 S 0.0 0.04.81 migration/0
4 root 34 19 0 0 0 S 0.0 0.14.61 ksoftirqd/0
5 root RT -5 0 0 0 S 0.0 0.00.00 watchdog/0
6 root RT -5 0 0 0 S 0.0 0.03.94 migration/1
7 root 34 19 0 0 0 S 0.0 0.03.86 ksoftirqd/1
8 root RT -5 0 0 0 S 0.0 0.00.00 watchdog/1
9 root 10 -5 0 0 0 S 0.0 0.00.03 events/0
10 root 10 -5 0 0 0 S 0.0 0.04.00 events/1
11 root 10 -5 0 0 0 S 0.0 0.00.02 khelper
31 root 12 -5 0 0 0 S 0.0 0.01.16 kblockd/0