WaveScript Benchmarks Performance Report

September 9, 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: 3563

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 mlton03 c2boehm c2boehmseglst c2 c2seglist c2def c2defseglst
just_timer 2568.000 2512.000 2516.000 2512.000 2528.000 5012.000 5024.000
readfile_bigwins 3788.000 448.000 1088.000 780.000 3908.000 260.000 1000.000
printing_lists 2640.000 932.000 900.000 848.000 824.000 840.000 796.000
conv_SigsegArr 2368.000 388.000 7288.000 816.000 5628.000 48.000 6664.000
fft 104.000 932.000 900.000 848.000 964.000 892.000 888.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4456.000
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 7140.000 11833.000 4164.000 7468.000 5604.000 7364.000 3820.000
## Running marmot2
test_marmot2 2336.000 5228.000 5260.000 4736.000 4672.000 4560.000 4548.000
## Running marmot3
test_heatmap 7748.000 3212.000 3232.000 2516.000 2508.000 3244.000 3240.000
## Running marmot multinode offline
run_3phases 9433.000 5876.000 4884.000 5620.000 5308.000 5512.000 4808.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 14:52:58 up 49 days, 23:27, 4 users, load average: 1.07, 1.06, 0.84
Tasks: 159 total, 1 running, 158 sleeping, 0 stopped, 0 zombie
Cpu(s): 28.2%us, 4.2%sy, 0.8%ni, 65.9%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1198752k used, 875204k free, 95708k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 747460k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
32643 newton 15 0 16836 114 4200 S 2 0.6 64:53.31 emcs21-x
1 root 18 0 2948 1856 532 S 0 0.1 0:05.57 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
3 root RT -5 0 0 0 S 0 0.0 0:00.44 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.86 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 0:00.34 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.19 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 19 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 12 -5 0 0 0 S 0 0.0 0:00.58 kblockd/0

Top results after running benchmarks:

top - 15:11:30 up 49 days, 23:45, 4 users, load average: 1.12, 1.31, 1.20
Tasks: 159 total, 1 running, 158 sleeping, 0 stopped, 0 zombie
Cpu(s): 28.2%us, 4.2%sy, 0.8%ni, 65.9%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1087364k used, 986592k free, 89168k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 66100k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:05.57 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 0:00.44 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:00.86 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 0:00.34 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:01.19 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 19 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 17 -5 0 0 0 S 0 0.0 0:00.58 kblockd/0
32 root 17 -5 0 0 0 S 0 0.0 0:00.00 kblockd/1