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: 3561

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  c2boehmseglist  c2  c2seglist  c2def  c2defseglist
just_timer  2556.000 2544.000 2512.000 2516.000 2516.000 5036.000 5016.000
readfile_bigwins  3792.000 440.000 1212.000 780.000 3856.000 264.000 1020.000
printing_lists  2776.000 440.000 1212.000 780.000 3856.000 264.000 1020.000
conv_SligsegArr  2352.000 424.000 7300.000 832.000 5420.000 56.000 6552.000
fft  104.000 876.000 936.000 872.000 892.000 764.000 840.000
```

#### Language Shootout:

```
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark   c2
fannkuch2  4444.000
```
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 7164.0 11801.000 4100.000 7380.000 5648.000 7152.000 3816.000

## Running marmot2
test_marmot2 2324.000 5304.000 5212.000 4692.000 4664.000 4544.000 4576.000

## Running marmot3
test_heatmap 7796.000 3228.000 3244.000 2540.000 2528.000 3220.000 3236.000

## Running marmot multinode offline
run_3phases 9409.000 5868.000 4876.000 5644.000 5292.000 5476.000 4804.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 11:56:07 up 49 days, 20:30, 4 users, load average: 0.99, 1.00, 0.84
Tasks: 155 total, 1 running, 154 sleeping, 0 stopped, 0 zombie
Cpu(s): 28.2%us, 4.2%sy, 0.8%ni, 65.8%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1436048k used, 637908k free, 111664k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 930776k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
28338 newton 20 0 2368 1080 784 R 2 0.1 0:00.01 top
1 root 18 0 2948 1586 532 S 0 0.1 0:05.52 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
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 watchd/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 watchd/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 12 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 34 19 0 0 0 S 0 0.0 0:00.58 kblockd/0

Top results after running benchmarks:

top - 12:14:20 up 49 days, 20:48, 4 users, load average: 1.03, 1.02, 0.96
Tasks: 156 total, 1 running, 155 sleeping, 0 stopped, 0 zombie
Cpu(s): 28.2%us, 4.2%sy, 0.8%ni, 65.8%id, 0.1%wa, 0.4%hi, 0.4%si, 0.0%st
Mem: 2073956k total, 1053764k used, 1020192k free, 107920k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 563200k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
32643 newton 15 0 15836 11m 4200 S 2 0.6 64:16.45 emacs21-x
1 root 15 0 2948 1586 532 S 0 0.1 0:05.52 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
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 watchd/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 watchd/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 12 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:00.58 kblockd/0

6