WaveScript Benchmarks Performance Report

October 20, 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: 3600

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.

![fannkuch benchmark chart](image)

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

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:  
## NOSUDO:  
## NICE:  
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer  2572.000  2552.000  2516.000  2564.000  2536.000  5036.000  5028.000  
readfile_bigwins  3804.000  464.000  1132.000  1228.000  3768.000  244.000  1032.000  
printing_lists  2736.000  900.000  904.000  868.000  840.000  848.000  848.000  
conv_SigsegArr  2308.000  364.000  7336.000  840.000  5676.000  20.000  6420.000  
fft  120.000  896.000  884.000  868.000  972.000  820.000  888.000  
```

Language Shootout:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:  
## NOSUDO:  
## NICE:  
Benchmark c2  
fannkuch2  4332.000  
```
Application Benchmarks:

## Real or User time for each benchmark/backend

## LD_PRELOAD:

## NOSUDO:

## NICE:

Benchmark mltonO3 c2boehm c2boehmseglst c2 c2seglst c2def c2defseglist

## Running orig marmot phase 1

run_first_phase 7140.000 11873.000 4068.000 7388.000 5556.000 7148.000 3756.000

## Running marmot2
test_marmot2 2328.000 5244.000 5252.000 4656.000 4672.000 4576.000 4580.000

## Running marmot3
test_heatmap 7748.000 3240.000 3236.000 2512.000 2536.000 3244.000 3224.000

## Running marmot multinode offline
run_3phases 9429.000 5924.000 4860.000 5588.000 5268.000 5472.000 4768.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 06:22:03 up 90 days, 14:56, 4 users, load average: 1.00, 1.04, 0.78
Tasks: 180 total, 1 running, 179 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.7%us, 3.1%sy, 0.8%ni, 67.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1575516k used, 498440k free, 150364k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 1061876k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
21875 newton 21 0 2488 1088 784 R 2 0.1 0:00.01 top
1 root 18 0 2948 1856 532 S 0 0.1 0:07.33 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 0:04.87 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.69 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 0:04.02 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:03.94 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 17 -5 0 0 0 S 0 0.0 0:01.47 kblockd/0

Top results after running benchmarks:

top - 06:40:58 up 90 days, 15:15, 4 users, load average: 1.00, 1.01, 0.93
Tasks: 180 total, 1 running, 179 sleeping, 0 stopped, 0 zombie
Cpu(s): 27.7%us, 3.1%sy, 0.8%ni, 67.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1043016k used, 1030940k free, 129156k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 580712k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:07.34 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 0:04.88 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.69 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 0:04.02 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:03.94 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 17 -5 0 0 0 S 0 0.0 0:01.47 kblockd/0