WaveScript Benchmarks Performance Report

September 16, 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: 3569

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 2512.000 2516.000 2512.000 2512.000 2524.000 5028.000 5036.000
readfile_bigwins 3772.000 472.000 1108.000 776.000 3892.000 236.000 1024.000
printing_lists 2404.000 904.000 900.000 852.000 876.000 812.000 828.000
conv_SigsegArr 2268.000 388.000 7464.000 824.000 5636.000 40.000 6696.000
fft 112.000 980.000 996.000 864.000 948.000 880.000 936.000

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 7360.000 12177.000 4324.000 7700.000 5620.000 8261.000 3880.000

## Running marmot2
test_marmot2 2324.000 5264.000 5240.000 4716.000 4664.000 4568.000 4560.000

## Running marmot3
test_heatmap 7768.000 3212.000 3204.000 2520.000 2508.000 3212.000 3236.000

## Running marmot multinode offline
run_3phases 9585.000 6148.000 5684.000 5284.000 5728.000 4788.000

B Appendix: Additional system information

Top results before running benchmarks:

```
top - 06:22:45 up 56 days, 14:56, 2 users, load average: 2.03, 2.03, 1.79
Tasks: 166 total, 3 running, 163 sleeping, 0 stopped, 0 zombie
Cpu(s): 29.7%us, 4.0%sy, 0.7%ni, 64.8%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem:  2073956k total, 1570840k used, 503116k free, 128908k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 1057308k cached

PID USER   PR NI VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16573 newton 25  0 34208  31m 492 R  99 1.6  7176:50 ikarus
  1 root    0  0  2948  1856  532 S   0  0.1  0:05.79 init
  2 root    0  0  0    0    0 S   0  0.0  0:00.00 kthreadd
  3 root    0  0  0    0    0 S   0  0.0  0:00.46 migration/0
  4 root    0  0  0    0    0 S   0  0.0  0:13.07 ksoftirqd/0
  5 root    0  0  0    0    0 S   0  0.0  0:00.00 watchd/0
  6 root    0  0  0    0    0 S   0  0.0  0:00.37 migration/1
  7 root    0  0  0    0    0 S   0  0.0  0:03.41 ksoftirqd/1
  8 root    0  0  0    0    0 S   0  0.0  0:00.00 watchd/1
  9 root    0  0  0    0    0 S   0  0.0  0:00.03 events/0
 10 root    0  0  0    0    0 S   0  0.0  0:00.04 events/1
 11 root    0  0  0    0    0 S   0  0.0  0:00.02 khelper
 31 root    0  0  0    0    0 S   0  0.0  0:00.66 kblockd/0

```

Top results after running benchmarks:

```
top - 06:41:46 up 56 days, 15:15, 2 users, load average: 1.93, 2.02, 1.94
Tasks: 166 total, 2 running, 164 sleeping, 0 stopped, 0 zombie
Cpu(s): 29.7%us, 4.0%sy, 0.7%ni, 64.8%id, 0.1%wa, 0.3%hi, 0.4%si, 0.0%st
Mem:  2073956k total, 1041324k used, 1032632k free, 131992k buffers
Swap: 14996668k total, 1041324k used, 14961924k free, 131992k buffers

PID USER   PR NI VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16573 newton 25  0 34208  31m 492 R  99 1.6  7195:45 ikarus
   1 root    0  0  2948  1856  532 S   0  0.1  0:05.79 init
   2 root    0  0  0    0    0 S   0  0.0  0:00.00 kthreadd
   3 root    0  0  0    0    0 S   0  0.0  0:00.46 migration/0
   4 root    0  0  0    0    0 S   0  0.0  0:13.07 ksoftirqd/0
   5 root    0  0  0    0    0 S   0  0.0  0:00.00 watchd/0
   6 root    0  0  0    0    0 S   0  0.0  0:00.37 migration/1
   7 root    0  0  0    0    0 S   0  0.0  0:03.41 ksoftirqd/1
   8 root    0  0  0    0    0 S   0  0.0  0:00.00 watchd/1
   9 root    0  0  0    0    0 S   0  0.0  0:00.03 events/0
  10 root    0  0  0    0    0 S   0  0.0  0:00.04 events/1
  11 root    0  0  0    0    0 S   0  0.0  0:00.02 khelper
  31 root    0  0  0    0    0 S   0  0.0  0:00.66 kblockd/0
```

---

6