WaveScript Benchmarks Performance Report

November 3, 2008

Machine information:
Linux chastity 2.6.22-14-generic #1 SMP Tue Feb 12 07:42:25 UTC 2008 i686 GNU/Linux
LD_PRELOAD: CC: cc

WaveScript SVN:
Revision: 3624

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.

3.1 Marmot Application

We start off by looking at the original, hand-optimized marmot application that we deployed. We break it down by phase: the first three phases of the computation, followed by all three together.
3.2 Computer Vision: Background Subtraction

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:
## CC:
## ‘which icc’ =
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2524.000 2544.000 2520.000 2540.000 2520.000 5036.000 5032.000
readfile_bigwins 3664.000 484.000 1156.000 1256.000 3812.000 244.000 932.000
printing_lists 2660.000 904.000 912.000 880.000 848.000 820.000 832.000
conv_SigsegArr 2308.000 420.000 7412.000 784.000 6268.000 44.000 6548.000
fft 112.000 960.000 884.000 836.000 880.000 888.000 872.000

Language Shootout:

## Real or User time for each benchmark/backend

## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## ‘which icc’ =
Benchmark c2
fannkuch2 4348.000

Application Benchmarks:

## Real or User time for each benchmark/backend

## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## ‘which icc’ =
Benchmark mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7388.000 11737.000 4176.000 7444.000 5560.000 7860.000 3940.000
## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2188.000 5260.000 5252.000 4648.000 4696.000 4528.000 4596.000
## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7748.000 3200.000 3220.000 2512.000 2524.000 3212.000 3216.000
B Appendix: Additional system information

Top results before running benchmarks:

```
Top - 11:11:14 up 104 days, 20:45, 6 users, load average: 2.10, 2.10, 1.62
Tasks: 203 total, 2 running, 201 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.6%us, 2.7%sy, 0.7%ni, 71.4%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 6933620k used, 1440584k free, 8624k buffers
Swap: 14996668k total, 624024k used, 14372644k free, 85240k cached

PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
17543 newton 25  0   76368  69m  2152   R   99  3.4  0:15.16 mzscheme
  1  root   15  0     88260 3044 1336   R  101  0.1  0:03.93 query.exe
  2  root   15  0     2946  1856  532   S   0.1  0:08.21 init
  3  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  4  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  5  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  6  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  7  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  8  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  9  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
 10  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
 11  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
 31  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
```

Top results after running benchmarks:

```
Top - 11:32:20 up 104 days, 21:06, 6 users, load average: 2.00, 2.04, 1.94
Tasks: 202 total, 2 running, 200 sleeping, 0 stopped, 0 zombie
Cpu(s): 24.6%us, 2.7%sy, 0.7%ni, 71.4%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 624024k used, 1449932k free, 8624k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 339516k cached

PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
  6627 newton 25  0   88260 3044 1336   R  101  0.1  0:03.93 query.exe
  1  root   15  0     2946  1856  532   S   0.1  0:08.21 init
  2  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  3  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  4  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  5  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  6  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  7  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  8  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
  9  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
 10  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
 11  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
 31  root   15  0     3419     0     0   S   0.0  0:00.00 kthread
```