WaveScript Benchmarks Performance Report

November 1, 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: 3623

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.

![Chart](chart.png)

### 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 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:
## CC:
## 'which icc' =
Benchmark mlton03 c2boehm c2boehmselist c2 c2seglst c2def c2defseglst
just_timer 2512.000 2532.000 2564.000 2520.000 2528.000 5032.000 5044.000
readfile_bigwins 3660.000 424.000 1108.000 1260.000 3776.000 252.000 844.000
printing_lists 2780.000 904.000 912.000 852.000 836.000 812.000 824.000
conv_SigsegArr 2336.000 412.000 7288.000 796.000 5568.000 40.000 6548.000
fft 140.000 940.000 924.000 888.000 976.000 888.000 924.000
```

Language Shootout:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' =
Benchmark c2
fannkuch2 4352.000
```

Application Benchmarks:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' =
Benchmark mlton03 c2boehm c2boehmselist c2 c2seglst c2def c2defseglst
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7448.000 12185.000 4152.000 7548.000 5672.000 8013.000 3984.000
## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2208.000 5264.000 5252.000 4632.000 4704.000 4512.000 4552.000
## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7748.000 3228.000 3216.000 2512.000 2512.000 3204.000 3228.000
```
Running benchmark marmot_all.bench for 20 tuples.
run_3phases 9505.000 5996.000 4864.000 5680.000 5332.000 5700.000 4788.000

Real or User time for each benchmark/backend
LD_PRELOAD:
NOSUDO:
NICE:
CC:

Benchmark mltonO3 c2boehm c2 c2def

Running benchmark bgsub3.bench for 10 tuples.
bgSub3_integer 10261.000 8469.000 9961.000 7800.000

Appendix: Additional system information

Top results before running benchmarks:
top - 11:09:09 up 102 days, 19:43, 7 users, load average: 0.99, 1.04, 0.80
Tasks: 198 total, 1 running, 197 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.0%us, 2.8%sy, 0.7%ni, 71.0%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1442480k used, 631476k free, 78684k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 1011952k cached

Top results after running benchmarks:
top - 11:29:46 up 102 days, 20:03, 7 users, load average: 2.16, 2.00, 1.50
Tasks: 207 total, 2 running, 205 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.0%us, 2.8%sy, 0.7%ni, 70.9%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1315216k used, 757740k free, 66168k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 810400k cached