WaveScript Benchmarks Perfomance Report

October 28, 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: 3619

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.

![Graph showing benchmark results](image)

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

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2564.000 2548.000 2528.000 2520.000 2556.000 5028.000 5032.000
readfile_bigwins 3664.000 444.000 1072.000 1280.000 3744.000 248.000 868.000
printing_lists 500_x_500_x_600reps
3_x_83333_x_600reps
fft 152.000 956.000 980.000 920.000 968.000 864.000 840.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4316.000

Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7340.000 11829.000 4072.000 7364.000 5528.000 7304.000 3996.000
## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2188.000 5232.000 5272.000 4652.000 4716.000 4560.000 4564.000
## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7732.000 3248.000 3212.000 2536.000 2504.000 3252.000 3244.000
## Running benchmark marmot_all.bench for 20 tuples.
run_3phases 9521.000 5884.000 5884.000 5652.000 5304.000 5528.000 4772.000
B Appendix: Additional system information

Top results before running benchmarks:

```
top - 15:25:09 up 98 days, 23:59, 6 users, load average: 1.01, 1.04, 0.85
Tasks: 196 total, 1 running, 195 sleeping, 0 stopped, 0 zombie
Cpu(s): 25.7%us, 2.9%sy, 0.8%ni, 70.1%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1442844k used, 631122k free, 71516k buffers
Swap: 14996668k total, 34748k used, 14961920k free, 965236k cached
```

Top results after running benchmarks:

```
27961 newton 21 0 2492 1100 784 R 2 0.1 0:00.01 top
32643 newton 15 0 15836 11m 4200 S 2 0.6 344:03.98 emacs21-x
```

---

### NICE:
Benchmark mltonO3 c2boehm c2 c2def
Running benchmark bgsub3.bench for 10 tuples.
bgSub3_integer 10209.000 8345.000 9833.000 7664.000