WaveScript Benchmarks Performance Report

November 12, 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: 3652

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 c2boehmseglst c2 c2seglist c2def c2defseglst
just_timer 2528.000 2512.000 2512.000 2512.000 2512.000 5036.000 5012.000
readfile_bigwins 3640.000 456.000 1040.000 1268.000 4572.000 252.000 928.000
printing_lists 2668.000 928.000 916.000 864.000 828.000 832.000 816.000
conv_SigsegArr 2260.000 400.000 7256.000 824.000 5276.000 24.000 6392.000
fft 140.000 904.000 892.000 872.000 1020.000 956.000 960.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## ‘which icc‘ =
Benchmark c2
fannkuch2 4512.000

Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## ‘which icc‘ =
Benchmark mltonO3 c2boehm c2boehmseglst c2 c2seglist c2def c2defseglst
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7624.000 12373.000 4272.000 7512.000 5716.000 9837.000 4008.000
## Running benchmark marmot2.bench for 150 tuples.
run_3phases 9497.000 5912.000 4912.000 5680.000 5284.000 5672.000 4872.000
## Real or User time for each benchmark/backend
B Appendix: Additional system information

Top results before running benchmarks:

Top results after running benchmarks: