WaveScript Benchmarks Performance Report

November 7, 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: 3634

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 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 mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2560.000 2520.000 2516.000 2536.000 2512.000 5012.000 5020.000
readfile_bigwins 3660.000 476.000 1120.000 1316.000 4528.000 252.000 820.000
printing_lists 2572.000 924.000 932.000 836.000 832.000 804.000 812.000
conv_SigsegArr 2320.000 324.000 7248.000 792.000 5300.000 84.000 6760.000
fft 116.000 900.000 860.000 936.000 852.000 880.000 864.000
```

Language Shootout:

```plaintext
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' =
Benchmark c2
fannkuch2 4516.000
```

Application Benchmarks:

```plaintext
## 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 7256.000 11905.000 4132.000 7444.000 5548.000 7504.000 4200.000
## Running benchmark marmot2.bench for 150 tuples.
test_marmot2 2216.000 5260.000 5244.000 4652.000 4664.000 4532.000 4576.000
## Running benchmark marmot3.bench for 14 tuples.
test_heatmap 7784.000 3268.000 3232.000 2504.000 2512.000 3228.000 3236.000
```
## Running benchmark marmot_all.bench for 20 tuples.

run_3phases 9521.000 5888.000 4836.000 5308.000 5636.000 4816.000

## Real or User time for each benchmark/backend

## LD_PRELOAD:

## NOSUDO:

## NICE:

## CC:

## ‘which icc’ =

Benchmark mltonO3 c2boehm c2 c2def

## Running benchmark bgsub3.bench for 10 tuples.

bgSub3_integer 10233.000 8349.000 9841.000 7672.000

B Appendix: Additional system information

Top results before running benchmarks:

top - 06:26:29 up 108 days, 16:00, 6 users, load average: 1.07, 1.62, 1.62
Tasks: 196 total, 1 running, 195 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.8%us, 2.7%sy, 0.7%ni, 72.3%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1384180k used, 689776k free, 91052k buffers
Swap: 14996668k total, 34744k used, 14961924k free, 877076k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1 root 18 0 2948 1856 532 S 0 0.1 0:08.40 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
3 root RT -5 0 0 0 S 0 0.0 0:05.71 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.79 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0 0.0 0:04.88 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:04.00 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0 0.0 0:01.78 events/0
10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
11 root 15 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:01.76 kblockd/0
32 root 10 -5 0 0 0 S 0 0.0 0:00.04 kblockd/1

Top results after running benchmarks:

top - 06:49:48 up 108 days, 16:23, 6 users, load average: 1.00, 1.02, 1.15
Tasks: 196 total, 1 running, 195 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.8%us, 2.7%sy, 0.7%ni, 72.3%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1385900k used, 1487056k free, 7856k buffers
Swap: 14996668k total, 34756k used, 14961912k free, 282152k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
11783 newton 19 0 2492 1100 784 R 2 0.1 0:00.01 top
1 root 18 0 2948 1856 532 S 0 0.1 0:08.40 init
2 root 11 -5 0 0 0 S 0 0.0 0:00.00 kthread
3 root RT -5 0 0 0 S 0 0.0 0:05.72 migration/0
4 root 34 19 0 0 0 S 0 0.0 0:14.79 ksoftirqd/0
5 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/0
6 root RT -5 0 0 0 S 0 0.0 0:04.88 migration/1
7 root 34 19 0 0 0 S 0 0.0 0:04.00 ksoftirqd/1
8 root RT -5 0 0 0 S 0 0.0 0:00.00 watchdog/1
9 root 10 -5 0 0 0 S 0 0.0 0:01.78 events/0
10 root 10 -5 0 0 0 S 0 0.0 0:00.04 events/1
11 root 15 -5 0 0 0 S 0 0.0 0:00.02 khelper
31 root 10 -5 0 0 0 S 0 0.0 0:01.76 kblockd/0

7