WaveScript Benchmarks Performance Report

November 11, 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: 3649

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

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## 'which icc' =
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2532.000 2544.000 2512.000 2540.000 2512.000 5024.000 5020.000
readfile_bigwins 3716.000 476.000 1104.000 1276.000 4640.000 256.000 820.000
printing_lists 2540.000 928.000 900.000 864.000 860.000 812.000 812.000
conv_SigsegArr 2232.000 376.000 7192.000 848.000 5424.000 60.000 6468.000
fft 124.000 856.000 920.000 928.000 916.000 844.000 888.000
```

Language Shootout:

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## 'which icc' =
Benchmark c2
fannkuch2 4504.000
```

Application Benchmarks:

```bash
## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## 'which icc' =
Benchmark mlton03 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7596.000 12361.000 4180.000 7620.000 5684.000 9901.000 4012.000
## Running benchmark marmot2.bench for 150 tuples.
run_3phases 9501.000 5872.000 4904.000 5652.000 5288.000 5952.000 4668.000
## 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 10373.000 8545.000 9977.000 8461.000

### Appendix: Additional system information

Top results before running benchmarks:

```
Top - 13:43:06 up 112 days, 23:17, 5 users, load average: 1.02, 1.01, 0.93
Tasks: 190 total, 1 running, 189 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.3%us, 2.0%sy, 0.7%ni, 72.9%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1337584k used, 736372k free, 15688k buffers
Swap: 1499668k total, 34748k used, 14961920k free, 1057316k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
3414 newton 21 0 2492 1096 784 R 2 0.1 0:00.01 top
  1 root 21 0 2948 1856 532 S 0 0.1 0:08.82 init
  2 root 10 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
  3 root RT -5 0 0 0 S 0 0.0 0:05.82 migration/0
  4 root 34 19 0 0 0 S 0 0.0 0:14.83 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 0:04.97 migration/1
  7 root 34 19 0 0 0 S 0 0.0 0:04.01 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 10 -5 0 0 0 S 0 0.0 0:00.02 khelper
 31 root 17 -5 0 0 0 S 0 0.0 0:01.83 kblockd/0
```

Top results after running benchmarks:

```
Top - 14:44:42 up 113 days, 18 min, 5 users, load average: 1.71, 1.87, 1.68
Tasks: 193 total, 2 running, 191 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.4%us, 2.6%sy, 0.7%ni, 72.9%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1256572k used, 817384k free, 18640k buffers
Swap: 1499668k total, 34748k used, 14961920k free, 970016k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
15863 root 18 0 10476 7668 1860 R 25 0.4 0:00.16 dpkg-preconfigu
 1 root 16 0 2948 1856 532 S 0 0.1 0:08.83 init
 2 root 10 -5 0 0 0 S 0 0.0 0:00.00 kthreadd
 3 root RT -5 0 0 0 S 0 0.0 0:05.82 migration/0
 4 root 34 19 0 0 0 S 0 0.0 0:14.83 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 0:04.98 migration/1
 7 root 34 19 0 0 0 S 0 0.0 0:04.01 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 10 -5 0 0 0 S 0 0.0 0:00.02 khelper
 31 root 17 -5 0 0 0 S 0 0.0 0:01.83 kblockd/0
```

7