WaveScript Benchmarks Performance Report

November 4, 2007

Machine information:
Linux faith 2.6.18-4-k7 #1 SMP Wed May 9 23:42:01 UTC 2007 i686 GNU/Linux

WaveScript SVN:
Revision: 2809

WaveScope Engine SVN:
Revision: 1495

1 Microbenchmarks

This section reports various microbenchmarks that stress the implementation of particular language constructs or data types.

![Microbenchmarks in all WS backends](image)

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.

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. Presently (2007.10) the largest of these by far is the marmot application.

3.1 Marmot Application

We start off by looking at the original, hand-optimized marmot application that we deployed.
4 Data Representation Profiling

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

## User time for each benchmark/backend
Benchmark "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFit DF -j 1" "MLton -O2"
readfile_bigwins 2320 1180 304.0 4.0 8.0 460.0

Language Shootout:

## User time for each language-shootout benchmark/backend
Benchmark "Scheme -O2" "Scheme -O3" "XStream -j 1 --at_once" "XStream DepthFirst -j 1 --at_once" "CoreFit DF -j 1" "MLton -O2"
fannkuch2 21290 16189 780.0 740.0 728.0 972.0

Marmot Application:

## Running original marmot app.
B Appendix: Additional system information

Top results before running benchmarks:

```
top - 11:53:06 up 97 days, 22:25, 14 users, load average: 2.61, 2.54, 2.41
Tasks: 485 total, 3 running, 482 sleeping, 0 stopped, 0 zombie
Cpu(s): 14.4%us, 1.7%sy, 3.2%ni, 78.9%id, 1.8%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 1155576k used, 920848k free, 113584k buffers
Swap: 1951856k total, 22940k used, 1728916k free, 699308k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
10786 newton 25 0 21308 18m 764 R 88 0.9 0:01.38 scheme
10820 newton 25 0 29928 20m 5612 R 32 1.0 0:00.18 cc1plus
10908 newton 18 0 2628 1296 792 R 2 0.1 0:00.03 top
1 root 15 0 2072 100 72 S 0 0.0 2:00.02 init
2 root RT 0 0 0 0 S 0 0.0 0:02.44 migration/0
3 root 34 19 0 0 0 S 0 0.0 0:35.66 ksoftirqd/0
4 root RT 0 0 0 0 S 0 0.0 0:02.97 migration/1
5 root 34 19 0 0 0 S 0 0.0 1:40.16 ksoftirqd/1
6 root 10 -5 0 0 0 S 0 0.0 0:00.16 events/0
7 root 10 -5 0 0 0 S 0 0.0 0:00.10 events/1
8 root 11 -5 0 0 0 S 0 0.0 0:00.00 khelper
9 root 10 -5 0 0 0 S 0 0.0 0:00.00 kthread
13 root 10 -5 0 0 0 S 0 0.0 0:00.62 kblockd/0
```

Top results after running benchmarks:

```
top - 14:11:41 up 93 days, 23:44, 12 users, load average: 0.25, 30.37, 205.91
Tasks: 453 total, 1 running, 449 sleeping, 3 stopped, 0 zombie
Cpu(s): 14.4%us, 1.3%sy, 3.4%ni, 80.0%id, 1.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2076424k total, 197428k used, 1878996k free, 6656k buffers
Swap: 1951856k total, 242288k used, 1709568k free, 64028k cached

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
10972 newton 15 0 57216 7504 5032 S 70 0.4 5:37.54 kpdf
12969 newton 16 0 20544 6040 228 D 16 0.3 0:25.94 sshd
23722 newton 15 0 2628 1280 792 R 4 0.1 0:00.03 top
23687 newton 15 0 8100 6644 1108 S 2 0.3 0:06.36 unison-2.28
1 root 15 0 2072 100 72 S 0 0.0 0:53.36 init
2 root RT 0 0 0 0 S 0 0.0 0:02.34 migration/0
3 root 34 19 0 0 0 S 0 0.0 0:28.10 ksoftirqd/0
4 root RT 0 0 0 0 S 0 0.0 0:02.71 migration/1
5 root 34 19 0 0 0 S 0 0.0 1:26.56 ksoftirqd/1
6 root 10 -5 0 0 0 S 0 0.0 0:00.07 events/0
7 root 10 -5 0 0 0 S 0 0.0 0:00.03 events/1
8 root 11 -5 0 0 0 S 0 0.0 0:00.00 khelper
9 root 10 -5 0 0 0 S 0 0.0 0:00.00 kthread
```