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: 3609

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. 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 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 mltonO3 c2boehm c2boehmseglist c2 c2seglist c2def c2defseglist
just_timer 2536.000 2528.000 2520.000 2524.000 2524.000 5024.000 5040.000
readfile_bigwins 3712.000 476.000 1164.000 1304.000 3764.000 268.000 1056.000
printing_lists 2744.000 884.000 904.000 844.000 888.000 824.000 828.000
conv_SigsegArr 2352.000 348.000 7664.000 776.000 5628.000 60.000 6356.000
fft 132.000 884.000 964.000 912.000 952.000 960.000 900.000

Language Shootout:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
Benchmark c2
fannkuch2 4336.000
**Application Benchmarks:**

### Real or User time for each benchmark/backend

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>mltonO3</th>
<th>c2boehm</th>
<th>c2boehmseglist</th>
<th>c2</th>
<th>c2def</th>
<th>c2defsseglist</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>run_first_phase</strong></td>
<td>7240.000</td>
<td>11921.000</td>
<td>4084.000</td>
<td>7332.000</td>
<td>5584.000</td>
<td>7192.000</td>
</tr>
<tr>
<td><strong>test_marmot2</strong></td>
<td>2192.000</td>
<td>5220.000</td>
<td>5236.000</td>
<td>4636.000</td>
<td>4722.000</td>
<td>4692.000</td>
</tr>
<tr>
<td><strong>test_heatmap</strong></td>
<td>7756.000</td>
<td>3266.000</td>
<td>3216.000</td>
<td>2508.000</td>
<td>2532.000</td>
<td>3244.000</td>
</tr>
<tr>
<td><strong>run_3phases</strong></td>
<td>9497.000</td>
<td>5908.000</td>
<td>4860.000</td>
<td>5616.000</td>
<td>5280.000</td>
<td>4880.000</td>
</tr>
<tr>
<td><strong>bgSub3_integer</strong></td>
<td>74984.000</td>
<td>6284.000</td>
<td>6268.000</td>
<td>6708.000</td>
<td>6720.000</td>
<td>6192.000</td>
</tr>
</tbody>
</table>

**B  Appendix: Additional system information**

Top results before running benchmarks:

```
top - 10:55:51 up 94 days, 19:30, 5 users, load average: 0.92, 1.00, 0.76
Tasks: 189 total, 1 running, 188 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.7%us, 3.0%sy, 0.8%ni, 69.0%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1611036k used, 462920k free, 87132k buffers
Swap: 1499668k total, 34752k used, 14961916k free, 1149948k cached

PID USER   PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+ COMMAND
1 root 21  0 2948 1856  532  S   0.1  0:07.64  init
2 root 13 -5  0   0   0  S   0.0  0:00.00  kthread
3 root RT -5  0   0   0  S   0.0  0:05.19  migration/0
4 root 34 19  0   0   0  S   0.0  0:14.72  ksoftirqd/0
5 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/0
6 root RT -5  0   0   0  S   0.0  0:04.37  migration/1
7 root 34 19  0   0   0  S   0.0  0:03.98  ksoftirqd/1
8 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/1
9 root 10 -5  0   0   0  S   0.0  0:01.78  events/0
10 root 10 -5  0   0   0  S   0.0  0:00.04  events/1
11 root 10 -5  0   0   0  S   0.0  0:00.02  khelper
31 root 10 -5  0   0   0  S   0.0  0:01.56  kblockd/0
32 root 11 -5  0   0   0  S   0.0  0:00.04  kblockd/1
```

Top results after running benchmarks:

```
top - 11:17:44 up 94 days, 19:51, 5 users, load average: 1.08, 1.04, 0.93
Tasks: 189 total, 1 running, 188 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.7%us, 3.0%sy, 0.8%ni, 69.0%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1405768k used, 668188k free, 91692k buffers
Swap: 1499668k total, 34752k used, 14961916k free, 957532k cached

PID USER   PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+ COMMAND
1 root 18  0 2948 1856  532  S   0.1  0:07.65  init
2 root 13 -5  0   0   0  S   0.0  0:00.00  kthread
3 root RT -5  0   0   0  S   0.0  0:05.20  migration/0
4 root 34 19  0   0   0  S   0.0  0:14.72  ksoftirqd/0
5 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/0
6 root RT -5  0   0   0  S   0.0  0:04.37  migration/1
7 root 34 19  0   0   0  S   0.0  0:03.98  ksoftirqd/1
8 root RT -5  0   0   0  S   0.0  0:00.00  watchdog/1
9 root 10 -5  0   0   0  S   0.0  0:01.78  events/0
10 root 10 -5  0   0   0  S   0.0  0:00.04  events/1
11 root 10 -5  0   0   0  S   0.0  0:00.02  khelper
31 root 10 -5  0   0   0  S   0.0  0:01.56  kblockd/0
32 root 11 -5  0   0   0  S   0.0  0:00.04  kblockd/1
```
31 root  10  -5  0  0  0  S  0  0.0  0:01.56 kblockd/0
32 root  10  -5  0  0  0  S  0  0.0  0:00.04 kblockd/1