WaveScript Benchmarks Performance Report

November 10, 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: 3645

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

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' =
Benchmark mltonO3 c2boehm c2boehmsegslist c2 c2segslist c2def c2defsegslist
just_timer 2564.000 2540.000 2516.000 2556.000 2516.000 5020.000 5024.000
readfile_bigwins 3692.000 452.000 1132.000 1276.000 4620.000 304.000 888.000
printing_lists 2664.000 884.000 892.000 844.000 872.000 808.000 824.000
conv_SigsegArr 2348.000 376.000 7376.000 840.000 5272.000 48.000 6456.000
fft 116.000 920.000 1000.000 904.000 968.000 900.000 832.000

Language Shootout:

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

Application Benchmarks:

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## 'which icc' =
Benchmark mltonO3 c2boehm c2boehmsegslist c2 c2segslist c2def c2defsegslist
## Running benchmark marmot1.bench for 100 tuples.
run_first_phase 7568.000 12401.000 4184.000 7612.000 5648.000 9833.000 3976.000
## Running benchmark marmot2.bench for 150 tuples.
run_3phases 9549.000 5888.000 4920.000 5688.000 5252.000 5916.000 4828.000

## Real or User time for each benchmark/backend
## LD_PRELOAD:
## NOSUDO:
## NICE:
## CC:
## `which icc` = Benchmark mltonO3 c2boehm c2 c2def

## Appendix: Additional system information

### Top results before running benchmarks:

```
top - 16:17:58 up 112 days, 1:52, 5 users, load average: 0.97, 1.00, 0.92
Tasks: 189 total, 1 running, 188 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.5%us, 2.6%sy, 0.7%ni, 72.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1255524k used, 818432k free, 110752k buffers
Swap: 14996668k total, 347432k used, 14961920k free, 829420k cached

PID USER   PR  NI  VIRT  RES  SHR  S  %CPU %MEM    TIME+  COMMAND
 1 root    20  0 2948 1856 532  S   0.0  0.1 0:08.74   init
 2 root    11 -5  0   0   0  532  S   0.0  0.0 0:00.00   kthreadd
 3 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 4 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 5 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 6 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 7 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 8 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 9 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
```

### Top results after running benchmarks:

```
top - 17:19:29 up 112 days, 2:53, 5 users, load average: 1.34, 1.80, 1.67
Tasks: 189 total, 1 running, 188 sleeping, 0 stopped, 0 zombie
Cpu(s): 23.5%us, 2.6%sy, 0.7%ni, 72.7%id, 0.2%wa, 0.2%hi, 0.2%si, 0.0%st
Mem: 2073956k total, 1057968k used, 1015968k free, 116812k buffers
Swap: 14996668k total, 347432k used, 14961920k free, 632996k cached

PID USER   PR  NI  VIRT  RES  SHR  S  %CPU %MEM    TIME+  COMMAND
 1 root    18  0 2948 1856 532  S   0.1  0.1 0:08.75   init
 2 root    11 -5  0   0   0  532  S   0.0  0.0 0:00.00   kthreadd
 3 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 4 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 5 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 6 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 7 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
 8 root    34 19  0   0   0  532  S   0.0  0.0 0:00.00   ksoftirqd/0
```