CoCoALib - Slug #742

View PP exponent vector (and order vector?) as an array of long?

28 Jun 2015 20:28 - John Abbott

Status:	In Progress	**Start date:**	28 Jun 2015
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	10%
Category:	Improving	**Estimated time:**	0.00 hour
Target version:	CoCoALib-1.0	**Spent time:**	1.20 hour

Description

Current implementations for exponent vectors (in PPMonoid elements) treat them as arrays of SmallExponent_t.

It might be faster to treat them as arrays of long so that several exponents can be added simultaneously -- JAA though we were doing this already.

Investigate; implement; measure; decide.

Related issues:

Related to CoCoALib - Slug #679: power for PPs is too slow	**Closed**	**13 Apr 2015**

History

#1 - 28 Jun 2015 20:31 - John Abbott

JAA believes that we are already using this trick for order vectors (managed by OrdvArith).

Is it worth doing the same for exponent vectors? Are there situations where CoCoALib would benefit appreciably?

#2 - 28 Jun 2015 20:35 - John Abbott

Here is a simplistic test program I have written. It should let you test whether any speed gain is likely if you view an array of machine integers as an array of long, for instance.

It creates C++ vector of long then views it also as an array of small (see typedef in the program). It times how long it takes to do a += operation.

```
#include "CoCoA/library.H"
using namespace CoCoA;
using namespace std;

void program()
{
  GlobalManager CoCoAFoundations;

  typedef long small;

  const int n = 3456;
  const int NumIters = 1000000;
  vector<long> buffer1(n);
  vector<long> buffer2(n, (256+1)*65536+256+1);

  double t0;

  // LOOP A "long"
  t0 = CpuTime();
  for (int i=0; i < NumIters; ++i)
    for (int j=0; j < n; ++j)
      buffer1[j] += buffer2[j];
  cout << "Time for long loop A: " << CpuTime()-t0 << endl;

  small* ptr1 = reinterpret_cast<small*>(&buffer1[0]);
  small* ptr2 = reinterpret_cast<small*>(&buffer2[0]);
  const int nn = n*(sizeof(long)/sizeof(small));

  // LOOP A "small"
  t0 = CpuTime();
```

```
  for (int i=0; i < NumIters; ++i)
    for (int j=0; j < nn; ++j)
      ptr1[j] += ptr2[j];
  cout << "Time for small loop A: " << CpuTime()-t0 << endl;

  // LOOP B "small"
  t0 = CpuTime();
  for (int i=0; i < NumIters; ++i)
  {
    const small* end = ptr1 + nn;
    small* p2 = ptr2;
    for (small* p1 = ptr1; p1 != end; ++p1, ++p2)
      *p1 += *p2;
  }
  cout << "Time for small loop B: " << CpuTime()-t0 << endl;

  // LOOP B "long"
  t0 = CpuTime();
  for (int i=0; i < NumIters; ++i)
    for (int j=0; j < n; ++j)
      buffer1[j] += buffer2[j];
  cout << "Time for long loop B: " << CpuTime()-t0 << endl;
}

//------------------------------------------------------------------------
// Use main() to handle any uncaught exceptions and warn the user about them.
int main()
{
  try
  {
    program();
    return 0;
  }
  catch (const CoCoA::ErrorInfo& err)
  {
    cerr << "***ERROR***  UNCAUGHT CoCoA error";
    ANNOUNCE(cerr, err);
  }
  catch (const std::exception& exc)
  {
    cerr << "***ERROR***  UNCAUGHT std::exception: " << exc.what() << endl;
  }
  catch(...)
  {
    cerr << "***ERROR***  UNCAUGHT UNKNOWN EXCEPTION" << endl;
  }

  CoCoA::BuildInfo::PrintAll(cerr);
  return 1;
}
```

#3 - 28 Jun 2015 20:43 - John Abbott

I get some strange timings on my computer (MacOSX-10.5.8, old compiler).

```
With small == long
Time for long loop A: 7.27442
Time for small loop A: 4.80132
Time for small loop B: 4.80288
Time for long loop B: 7.32373

With small == int
Time for long loop A: 7.26863
Time for small loop A: 7.08082
Time for small loop B: 7.08475
Time for long loop B: 7.30074

With small == short
Time for long loop A: 7.27959
Time for small loop A: 12.3736
Time for small loop B: 12.3757
Time for long loop B: 7.29765

With small == unsigned char
Time for long loop A: 7.28086
Time for small loop A: 23.9722
Time for small loop B: 23.9661
Time for long loop B: 7.29536
```

On my platform it does seem beneficial to view the array as being composed of long; it seems to make no difference if I say signed or unsigned.

NOTE it is faster to view the exponents as a C array of long rather than as a C++ vector<long>; I'm hoping this is because I'm using a crummy old compiler (Apple gcc-4.2)

NOTE2 it is faster still running in a virtual Linux machine on my portable; so I guess the Apple compiler is at fault for the strange timings.

#4 - 29 Jun 2015 15:42 - John Abbott

- Status changed from New to In Progress

- % Done changed from 0 to 10

On my 32-bit linux virtual machine I observe the following:

- with int and long all loops take about 3.0s (NB int and long are the same size)
- with short the "small" loops take about 6.0s
- with char the "small" loops take about 11.9s

NOTE compiler is gcc-4.8.2

#5 - 01 Sep 2015 11:30 - John Abbott

Could someone else try to run the test program on a decent machine with a recent compiler?

My conclusion from the Linux VM test is that there is a significant gain from viewing a C array of int (or short) as an array of long provided you use a modern compiler. I hope someone else can confirm this... Christof?

If so, we should modify our code to use this approach (at least when debugging is off).

#6 - 01 Sep 2015 12:08 - Christof Soeger

For me all the timings are the same:

```
Time for long loop A: 3.59898
Time for small loop A: 3.60494
Time for small loop B: 3.61403
Time for long loop B: 3.62262
```

NOTE Christof's compiler is gcc-4.8.4