

Re: what's it worth to write a short program for polynomial multiplication?

Re: what's it worth to write a short program for polynomial multiplication?

Source: <http://sci.tech-archive.net/Archive/sci.math.symbolic/2008-06/msg00003.html>

- *From:* hrrubin@xxxxxxxxxxxxxxxxxxxxxx (Herman Rubin)
 - *Date:* 2 Jun 2008 16:04:55 -0400
-

In article <1b395e29-f3d3-4851-aeb8-f2f136a12ad7@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>, rjf <fateman@xxxxxxxx> wrote:

On Jun 2, 8:39 am, hru...@xxxxxxxxxxxxxxxxxxxxxx (Herman Rubin) wrote:

In article <570bb6f4-6c19-42f1-bc82-f6872d4f6...@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>,

Well, the next sentence in the abstract is...

What is the smallest expression in a programming language for such a program? (Assuming that the language does not have "multiplication of polynomials" as a primitive!)

HR:

Obviously, in an appropriate programming language, this

will be a basic operation, so the smallest expression would be

$$z = x * y,$$

Re: what's it worth to write a short program for polynomial multiplication?

where the overloaded operator * does what is expected.

This is not really a solution unless the overloaded operator is a lot smarter than we usually expect.

I agree that the overloaded operator is rarely capable of this. But this is the fault of the languages; there is no reason why it should not be.

That is, "*" must not only look at the types of x and y [which are presumably "polynomial"] but at other information, such as the domain of the coefficients, which could be integers, floats, intervals, matrices, elements in a finite field, strings, etc. It might also look at the sizes, density, number of variables, and it might look at the environment: how many processors are available, how much memory, etc. It certainly should look at both x and y, and perhaps also look at how z is described.

Alas, it is unlikely that the superfast subimbecile of a computer will be able to do this at run time; even at compile time, it is likely that the user will have to tell the compiler what is provided and what is wanted. Sparse polynomials and dense polynomials should be handled differently, and it is usually the user who can provide information on what is sparse and what is dense.

There is, I think, substantial evidence that the notion of object-oriented choice of operator based on one operand, and in the typical shallow way offered by most OO programming languages, is rather weak for helping in determining the proper algorithm.

Why should not the object-oriented choice of operator use the types of all the operands?

If there is exactly one algorithm for multiplying polynomials that blindly pushes ahead oblivious to whether the inputs are (say) sparse or dense, then you can be sure that the algorithm is going to be very inefficient on certain inputs.

Few of the current programming languages come even close to what is needed for good mathematics, and it

Re: what's it worth to write a short program for polynomial multiplication?

Re: what's it worth to write a short program for polynomial multiplication?

might be a good idea for mathematicians who can understand the problems to put such languages together,

This was the motivation for several programs of study in computer algebra systems, and resulted in several system designs, including most notably, Axiom/ Aldor.

I have observed that mathematicians, as well as physicists, may not be as good as they think they are in designing computer programming languages.

Has there been a real effort on this? Getting input from all on what is wanted, and then trying to do as well as one can.

A good assembler program, in a good assembler language, is less than twice the length of a HLL program which does not have strong intermediate operation in it. See the above smallest expression.

I'm not sure how one could judge this; but on simple size, I'm not so convinced.

Again, has anyone tried to produce an assembler which can handle weakly typed arguments with type overrides? I tried it on two machines, the CYBER 205 and the VAX 780, and came up with rather simple procedures. These ideas will also work on the POWER machines and related one.

What I did was to look at the machine instructions from a mathematical point of view and construct a simple grammar from the lot. Seymour Cray's assembler languages are not bad, but he did not try overloading.

If someone is interested in working with me to produce such, I would be willing to cooperate, but I am not in a CS environment myself. As I said, I would like the assembler developed to be close to as easy to use as a HLL when that is good at the job, and not too bad anywhere. As it would be overloaded, the short step I indicated might even be there; what one would not have is more

Re: what's it worth to write a short program for polynomial multiplication?

Re: what's it worth to write a short program for polynomial multiplication?

complicated expressions involving more calls. The "*" operator can have its forms on dense polynomials, sparse polynomials, etc. When I say overloaded, I mean overloaded. The user gives the instruction the types, and can override it when needed.

In the paper I indicated, one can represent sparse multivariate polynomials as hash tables whose indexes are lists of exponents of the different variables, and whose entries are the coefficients. Then producing a product hashtable can be done this way:

```
(defun ptimes(r s)
  (let ((ans (make-hash-table :test 'equal )))
    (maphash #'(lambda(ex co) ; exponent, coefficient
      (maphash #'(lambda (ex2 co2)
        (incf (gethash (mapcar #' + ex ex2) ans
          0)
          (* co co2)))) r) s)
    ans))
```

This is 7 lines of Common Lisp, though since the language is free-form, it could be printed all on one line :)
The result can be converted from a hashtable to a list in a few lines.

This is definitely not a recommended program for (say) dense, univariate polynomials, which might look more like this below, using a different data representation of lists of coefficients.
This program accumulates the answer in an array, and then converts to a list.

```
(defun ptimes (L1 L2)(array2list (ptimesA2 L1 L2)))
```

```
(defun ptimesA2(L1 L2)
  (if (or (null L1)(null L2)) nil
      (let ((ans (make-array (+ 1(cdar L1)(cdar L2)) :initial-element 0)))
```

Re: what's it worth to write a short program for polynomial multiplication?

Re: what's it worth to write a short program for polynomial multiplication?

```
(dolist (i L1 ans)
  (dolist (j L2)
    (incf (aref ans (+ (cdr i)(cdr j))) (* (car i)(car j))))))
```

```
(defun array2list(A) ; 30x^10+5x^3 is ((30 . 10)(5 . 3))
  (let ((ans nil))
    (dotimes (i (length A) ans)
      (unless (= 0 (aref A i)) (push (cons (aref A i) ans))))))
```

I don't pretend that these are totally obvious to the reader; they are intended to be fairly ordinary common lisp, though. I invite versions in (say) Python or other languages. Note that most common lisp implementations allow you to add type declarations and compile to pretty good machine language. You can also profile the program and then decide if it is worthwhile to insert assembler somewhere.

—
This address is for information only. I do not claim that these views are those of the Statistics Department or of Purdue University.
Herman Rubin, Department of Statistics, Purdue University
hrubin@xxxxxxxxxxxxxxxxx Phone: (765)494-6054 FAX: (765)494-0558
.