Subject: Optimising A = B+C?
Posted by kgb on Fri, 31 May 1996 07:00:00 GMT
View Forum Message <> Reply to Message

Does anyone know how IDL optimises A=B+C where
A,B and C are arrays?

I did a test a while ago and it was several times faster
than C code along the lines of:

```
i=n;
while (i--)
   *a-- = *b-- + *c--
```

(This was on a 2048x2048 array and everything fiited into
physical memory.)

Is it just that it is done in assembler or something?

just curious...

Karl
--
kgb@aaoepp.aao.gov.au   [Anglo-Australian Observatory]
----> pubs: http://www.ast.cam.ac.uk/~kgb/papers.html
----> pgperl: http://www.ast.cam.ac.uk/~kgb/pgperl.html

Subject: Re: Optimising A = B+C?
Posted by davis on Sun, 02 Jun 1996 07:00:00 GMT
View Forum Message <> Reply to Message

On Fri, 31 May 1996 02:24:03 GMT, Karl Glazebrook <kgb@aaoepp.aao.gov.au>
wrote:
: Does anyone know how IDL optimises A=B+C where
: A,B and C are arrays?
:
: I did a test a while ago and it was several times faster
: than C code along the lines of:
:
: i=n;
: while (i--)
:   *a-- = *b-- + *c--

Although I have no proof, I believe that the above can be coded to run
faster, e.g.,

```
  for (i = 0; i < n; i++) a[i] = b[i] + c[i];
```

Also, if IDL arrays are allocated with an even number of elements (with the last one padded for odd sized arrays), this is faster:

```
  for (i = 0; i < n; i++)
    {
      a[i] = b[i] + c[i];
      i++;
      a[i] = b[i] + c[i];
    }
```

:
: (This was on a 2048x2048 array and everything fiited into
: physical memory.)
:

It also depends upon how your two-dimensional array was implemented. One common C approach is to use, e.g.,

```
  double **a;
  unsigned int i;

  a = malloc (2048 * sizeof (double *));
  for (i = 0; i < 2048; i++)
    a[i] = malloc (2048 * sizeof(double));
```

Now consider adding such arrays together. One might code this as

```
  for (i = 0; i < 2048; i++)
    {
       for (j = 0; j < 2048; j++)
  {
     a[i][j] = b[i][j] + c[i][j];
  }
    }
```

I am not sure how many compilers will optimize this. For that reason, I never code loops like the above. Instead, I always do

```
  for (i = 0; i < 2048; i++)
    {
       double *ai, *bi, *ci;
 ai = a[i]; bi = b[i]; ci = c[i];
       for (j = 0; j < 2048; j++)
  {
     ai[j] = bi[j] + ci[j];
  }
```

```
    }
```

Finally, if you simply create such arrays via:

```
    double a[2048][2048];  /* this is just a 2048*2048 block of doubles */
```

then you might be tempted to perform the addition as

```
   for (i = 0; i < 2048; i++)
     {
        for (j = 0; j < 2048; j++)
   {
      a[i][j] = b[i][j] + c[i][j];
   }
     }
```

The problem with this is that it is not very friendly to your CPU cache
because the loop over j does not deal with neighboring values in the array.
As a result, it is best to calculate the sum for these types of arrays as:

```
   double *aa, *bb, *cc;
   unsigned int i;

   aa = (double *) a;
   bb = (double *) b;
   cc = (double *) c;

   for (i = 0; i < 2048*2048; i++)
     {
        aa[i] = bb[i] + cc[i];
     }
```

--
John E. Davis   Center for Space Research/AXAF Science Center
617-258-8119   MIT 37-662c, Cambridge, MA 02139
http://space.mit.edu/~davis

Subject: Re: Optimising A = B+C?
Posted by davis on Mon, 03 Jun 1996 07:00:00 GMT

View Forum Message <> Reply to Message

On 2 Jun 1996 16:53:52 GMT, John E. Davis <davis@space.mit.edu>
wrote:
: On Fri, 31 May 1996 02:24:03 GMT, Karl Glazebrook <kgb@aaoepp.aao.gov.au>
: wrote:
: : Does anyone know how IDL optimises A=B+C where
: : A,B and C are arrays?

```
: Also, if IDL arrays are allocated with an even number of elements (with the
: last one padded for odd sized arrays), this is faster:
:
:    for (i = 0; i < n; i++)
:      {
:        a[i] = b[i] + c[i];
:        i++;
:        a[i] = b[i] + c[i];
:      }
```

This really does appear to what IDL does. For example, below is an IDL procedure and equivalent C code:

```
pro test, nloops, val
  a = make_array (1024, 1024, /FLOAT, value=0)
  b = make_array (1024, 1024, /FLOAT, value=val)
  for i=1,nloops do begin
    a = a + b
    print, a(10, 10)
  endfor
end
```

The C code follows at the end of this message. Here is the result of simply compiling the C code and running it:

```
Script started on Mon Jun  3 17:49:46 1996

# acc -O t.c
# time ./a.out 10 1.3
1.300000
2.600000
3.900000
5.200000
6.500000
7.800000
9.100000
10.400001
11.700001
13.000001
18.848u 1.079s 0:20.03 99.4% 0+4061k 0+0io 0pf+0w
```

It turns out that this is much slower than the IDL routine executes. However, if is is compiled to use extra padding in arrays, it runs 3 times faster:

```
# acc -DEXTRA=1 -O t.c
# time ./a.out 10 1.3
1.300000
```

2.600000
3.900000
5.200000
6.500000
7.800000
9.100000
10.400001
11.700001
13.000001
6.679u 1.109s 0:07.95 97.7% 0+3793k 0+0io 0pf+0w

acc -DEXTRA=2 -O t.c
time ./a.out 10 1.3
1.300000
2.600000
3.900000
5.200000
6.500000
7.800000
9.100000
10.400001
11.700001
13.000001
6.329u 1.119s 0:07.60 97.7% 0+3764k 0+0io 0pf+0w

acc -DEXTRA=8 -O t.c
time ./a.out 10 1.3
1.300000
2.600000
3.900000
5.200000
6.500000
7.800000
9.100000
10.400001
11.700001
13.000001
5.099u 0.969s 0:06.19 97.7% 0+3656k 0+0io 0pf+0w

exit

exit

script done on Mon Jun 3 17:53:43 1996

Here is the file t.c:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 1024
#ifndef EXTRA
# define EXTRA 0
#endif
int main (int argc, char **argv)
{
   unsigned int dim = MAX_SIZE * MAX_SIZE;
   unsigned int i;
   float a[MAX_SIZE+EXTRA][MAX_SIZE+EXTRA], b[MAX_SIZE+EXTRA][MAX_SIZE+EXTRA];
   float *ap, *bp;
   int nloops, loop;
   float val;

   if ((argc != 3)
       || (1 != sscanf (argv[1], "%d", &nloops))
       || (1 != sscanf (argv[2], "%f", &val)))
     {
 fprintf (stderr, "Usage: %s NLOOPS VALUE\n",
   argv[0]);
 return 1;
     }


   ap = (float *) a;
   bp = (float *) b;

   for (i = 0; i < dim; i++)
     {
 ap[i] = 0.0;
 bp[i] = val;
#if EXTRA > 0
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
#if EXTRA > 1
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
#if EXTRA > 2
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
```

```
#if EXTRA > 3
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
#if EXTRA > 4
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
#if EXTRA > 5
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
#if EXTRA > 6
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
#if EXTRA > 7
 i++;
 ap[i] = 0.0;
 bp[i] = val;
#endif
     }

   for (loop = 0; loop < nloops; loop++)
     {
 for (i = 0; i < dim; i++)
   {
     ap[i] += bp[i];
#if EXTRA > 0
     i++;
     ap[i] += bp[i];
#endif
#if EXTRA > 1
     i++;
     ap[i] += bp[i];
#endif
#if EXTRA > 2
     i++;
     ap[i] += bp[i];
#endif
#if EXTRA > 3
     i++;
     ap[i] += bp[i];
#endif
```

```
#if EXTRA > 4
      i++;
      ap[i] += bp[i];
#endif
#if EXTRA > 5
      i++;
      ap[i] += bp[i];
#endif
#if EXTRA > 6
      i++;
      ap[i] += bp[i];
#endif
#if EXTRA > 7
      i++;
      ap[i] += bp[i];
#endif
   }
 fprintf (stdout, "%f\n", a[9][9]);
     }

   return 0;
}
```

--
John E. Davis   Center for Space Research/AXAF Science Center
617-258-8119   MIT 37-662c, Cambridge, MA 02139
http://space.mit.edu/~davis

Subject: Re: Optimising A = B+C?
Posted by davis on Tue, 04 Jun 1996 07:00:00 GMT
View Forum Message <> Reply to Message

On 3 Jun 1996 22:05:03 GMT, John E. Davis <davis@space.mit.edu>
wrote:
 : Here is the file t.c:
[...]
 : int main (int argc, char **argv)
 : {
 :   unsigned int dim = MAX_SIZE * MAX_SIZE;
 :   unsigned int i;
 :   float a[MAX_SIZE+EXTRA][MAX_SIZE+EXTRA],
b[MAX_SIZE+EXTRA][MAX_SIZE+EXTRA];

On some systems it is necessary to make `a' and `b' global variables because
as declared, they use up about 2*4*1024*1024 = 8 Mbytes of stack space.
This will cause the program to core dump if not enough stack space is
available. The change will look like:

```
float a[MAX_SIZE+EXTRA][MAX_SIZE+EXTRA], b[MAX_SIZE+EXTRA][MAX_SIZE+EXTRA];
int main (int argc, char **argv)
{
  unsigned int dim = MAX_SIZE * MAX_SIZE;
  unsigned int i;
```

--
John E. Davis   Center for Space Research/AXAF Science Center
617-258-8119   MIT 37-662c, Cambridge, MA 02139
http://space.mit.edu/~davis