HW6 Solutions
Computational Physics
Spring 2004

Gregg Stiesberg

4/4/04

Optimization

On a PIII 1GHz, with Itmx=10, broyden took 8.1 seconds.

After replacing dvector calls in rkp480(), broyden takes only 4.5 seconds, half the original time.

I initially had problems with gprof. To get the function call information in the output. I added explicit rules for p480, nrutil and physics in the makefile, adding -pg -O3 for each:

```
broyden_gprof: broyden.c $(OBJECTS) $(HEADERS)
cc -pg -O3 -c -o p480.o p480.c
cc -pg -O3 -c -o nrutil.o nrutil.c
cc -pg -O3 -c -o physics.o physics.c
cc -o broyden -pg -O3 -fprofile-arcs broyden.c $(OBJECTS) $(LIBRARIES)
```

Thanks to Prof. Arias for the help getting this to compile properly. This made the call information appear in gprof.

This also reduced the runtime from 4.4 to 2.9 seconds!

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
58.77	1.51	1.51	6628000	0.00	0.00	rk4p480Sch
19.07	2.00	0.49	3389	0.14	0.59	schint
14.79	2.38	0.38	25	15.20	98.82	getg
1.95	2.43	0.05	75	0.67	1.93	getPsi
1.95	2.48	0.05				free_f3tensor
1.95	2.53	0.05				main
0.78	2.55	0.02	50000	0.00	0.00	rk4p480
0.39	2.56	0.01	125	0.08	0.08	simpint
0.39	2.57	0.01	25	0.40	1.28	getphi
0.00	2.57	0.00	150000	0.00	0.00	derivs_Poisson
0.00	2.57	0.00	10406	0.00	0.00	dvector

```
  0.00      2.57     0.00    10406     0.00     0.00  free_dvector
  0.00      2.57     0.00     2614     0.00     0.59  func_Schrodinger
  0.00      2.57     0.00      625     0.00     0.59  func_SchrodingerNodes
  0.00      2.57     0.00      150     0.00     2.46  rtbisp480
  0.00      2.57     0.00      100     0.00     0.00  interpolate
  0.00      2.57     0.00       75     0.00    20.57  zriddrp480
  0.00      2.57     0.00       30     0.00     0.00  dmatrix
  0.00      2.57     0.00       30     0.00     0.00  free_dmatrix
  0.00      2.57     0.00       25     0.00     0.00  d3tensor
  0.00      2.57     0.00       25     0.00     0.00  free_d3tensor
  0.00      2.57     0.00       10     0.00     0.00  lubksbp480
  0.00      2.57     0.00       10     0.00     0.00  ludcmpp480
  0.00      2.57     0.00        2     0.00     0.00  ivector
  0.00      2.57     0.00        1     0.00     0.00  free_ivector
```

For some reason rk4p480Sch is not being inlined.

Timing

Note: the following results are from a PIII 1GHz 256k cache, 512MB RAM, running Debian Sarge with many other processes in background. Your mileage may vary.

Inner Loop Operations

The output of the various timer() calls:

```
**** n=256 ****
               mult:  195 MFLOPS  = 16.8 MFLOP / 0.086 sec
             divide:   25 MFLOPS  = 16.8 MFLOP / 0.665 sec
               sub :  194 MFLOPS  = 16.8 MFLOP / 0.086 sec
                if :    3 MFLOPS  = 16.8 MFLOP / 5.931 sec
              alloc:    5 MFLOPS  = 16.8 MFLOP / 3.682 sec
            multadd:  215 MFLOPS  = 33.6 MFLOP / 0.156 sec
              cmat1:   24 MFLOPS  = 16.8 MFLOP / 0.692 sec
              cmat2:   17 MFLOPS  = 16.8 MFLOP / 0.971 sec
Notes:
Division      costs  7.7 mults
Subroutine    costs  0.0 mults
If            costs 67.9 mults
malloc        costs 41.8 mults
Memory access costs  7.0 mults
Cache miss    costs  1.4 accesses
```

Clearly the compiler inlined the sub() routine, even though I put it in a separate file. I would not expect the If() to eat so many clock cycles; your results should

be lower for this. One common mistake here was forgetting to change the n^3 to n^2 in the print statement also.

BLAS2

```
**** n=1024 ****
             vecmat1: 77793 MFLOPS  = 2147.5 MFLOP / 0.028 sec
             vecmat2: 115643 MFLOPS  = 2147.5 MFLOP / 0.019 sec
**** n=2048 ****
             vecmat1: 125060 MFLOPS  = 17179.9 MFLOP / 0.137 sec
             vecmat2: 238523 MFLOPS  = 17179.9 MFLOP / 0.072 sec
**** n=4096 ****
             vecmat1: 50878 MFLOPS  = 137439.0 MFLOP / 2.701 sec
             vecmat2: 478816 MFLOPS  = 137439.0 MFLOP / 0.287 sec
```

The speed advantage is more dramatic with larger matrices.

BLAS3

With Nb=32:

```
**** n=128 ****
             matmat1:  308 MFLOPS  = 4.2 MFLOP / 0.014 sec
             matmat2:  320 MFLOPS  = 4.2 MFLOP / 0.013 sec

IMPORTANT: Test of matmat2; should be near zero... 1.525881e-05
**** n=256 ****
             matmat1:   47 MFLOPS  = 33.6 MFLOP / 0.710 sec
             matmat2:  305 MFLOPS  = 33.6 MFLOP / 0.110 sec

IMPORTANT: Test of matmat2; should be near zero... 3.814698e-06
**** n=512 ****
             matmat1:   48 MFLOPS  = 268.4 MFLOP / 5.638 sec
             matmat2:  119 MFLOPS  = 268.4 MFLOP / 2.263 sec

IMPORTANT: Test of matmat2; should be near zero... 9.536744e-07
**** n=1024 ****
             matmat1:   47 MFLOPS  = 2147.5 MFLOP / 45.449 sec
             matmat2:   28 MFLOPS  = 2147.5 MFLOP / 76.014 sec

IMPORTANT: Test of matmat2; should be near zero... 2.384186e-07
```

For small enough matrices, the methods are both able to take advantage of cache. For intermediate sizes, the block method clearly wins, and for large matrices the block method is actually slower.

LAPACK

Here I had to switch to the lab machines, as I don't have the fortran libs at home (I use Blitz++ and cblas instead). Using the suggested compile line, but adding -o lapack, the results were

```
Numrec:  89 MFLOP in 3.640 sec ==> 25 MFLOPS.
LAPACK:  89 MFLOP in 0.304 sec ==> 295 MFLOPS.

Numerical Recipes rms error: 1.554424e-10
LAPACK            rms error: 6.567748e-10
```

which is an order of magnitude improvement! Lesson for the day: Use blas (or cblas) for large in-loop matrix operations, and use the NR or Blitz or (your favorite package) for outside loop and/or setting up operations.

Octave

```
N = 100
dt=0.192347 MFLOPS=5
ans = 21
dt=0.004650 MFLOPS=143
ans = 23
N = 200
dt=0.012988 MFLOPS=615
ans = 23
dt=0.013557 MFLOPS=393
ans = 23
N = 300
dt=0.040874 MFLOPS=660
ans = 23
dt=0.040330 MFLOPS=446
ans = 23
N = 400
dt=0.095459 MFLOPS=670
ans = 23
dt=0.080473 MFLOPS=530
ans = 23
N = 500
dt=0.185065 MFLOPS=675
ans = 23
dt=0.139197 MFLOPS=598
ans = 23
N = 512
dt=0.210530 MFLOPS=637
ans = 23
```

```
dt=0.165655 MFLOPS=540
ans = 23
N = 600
dt=0.298311 MFLOPS=724
ans = 23
dt=0.221109 MFLOPS=651
ans = 23
N = 700
dt=0.469644 MFLOPS=730
ans = 23
dt=0.319989 MFLOPS=714
ans = 23
N = 800
dt=0.712179 MFLOPS=718
ans = 23
dt=0.451283 MFLOPS=756
ans = 23
N = 900
dt=1.002465 MFLOPS=727
ans = 23
dt=0.609142 MFLOPS=797
ans = 23
N = 1000
dt=1.376309 MFLOPS=726
ans = 23
dt=0.833393 MFLOPS=799
ans = 23
```

This is about half what we'd expect of the lab machines; there were in fact many jobs running on the lab machines, likely using half the processor time.