Question 4: Profiling

Sean Marshallsay: sm1183

The `simple_md.c` program was compiled with no optimisations and timed five times with the C-Shell `time` command, the timings were:

User Time (s)	System Time (s)	Elapsed Time (s)	CPU Usage
68.515	0.178	68.83	99.7%
69.940u	0.151s	70.18	99.8%
68.186u	0.105s	68.34	99.9%
69.129u	0.156s	69.39	99.8%
64.415u	0.023s	64.44	99.9%

and the results were:

PE	KE	Error
49854482.95817039161920547485	0.00000000000000000000	0.00000000000000000000
49854482.95817039161920547485	0.00243672236278765072	0.00000000004887664439

The same was done with compiler optimisations on:

User Time (s)	System Time (s)	Elapsed Time (s)	CPU Usage
12.341	0.013	12.36	99.9%
12.396	0.012	12.41	99.9%
11.978	0.008	11.99	99.8%
12.038	0.006	12.04	99.9%
12.068	0.007	12.08	99.8%

PE	KE	Error
49854482.95817039161920547485	0.00000000000000000000	0.00000000000000000000
49854482.95817039161920547485	0.00243672236278765072	0.00000000004887664439

The optimised version runs in about a sixth of the time. Both versions of the program have almost complete CPU utilisation so the speed-ups are due to optimisations in the code (rather than other a smaller load on the CPU). Both versions also gave identical results.

Running `gprof` on the unoptimised version showed that ∼60% of the runtime was spent in the `compute()` function, of this over half the runtime was spent in `dv()` and `v()`. `v()` was changed to avoid calling `pow()` as follows:

```
double v(double x){
  double sin_min = sin(min(x, pi_2));
  return sin_min*sin_min;
}
```

This reduced the runtime of the unoptimised version to about 65 seconds but had no affect on the optimised version. `dv()` was then changed to use an addition instead of multiplication by two:

```
double dv(double x){
  double sin_cos = sin(min(x,pi_2))*cos(min(x,pi_2));
  return sin_cos+sin_cos;
}
```

This had little effect on the runtime, however it was noticed that `min()` was taking up 25% of the total runtime and accounted for the majority of the time spent in `dv()` so `dv()` was changed to only call `min()` once:

```
double dv(double x){
  double x_pi_min = min(x,pi_2);
  double sin_cos = sin(x_pi_min)*cos(x_pi_min);
  return sin_cos+sin_cos;
}
```

This reduced the runtime of the unoptimised version to about 58 seconds but again had no effect on the optimised version. The results after these changes were unchanged:

PE	KE	Error
49854482.95817039161920547485	0.00000000000000000000	0.00000000000000000000
49854482.95817039161920547485	0.00243672236278765072	0.00000000004887664439

The program was profiled again after these changes. The amount of time spent in `min()` had been reduced to about 18% but `compute()` still dominated the runtime (about 68%). Within compute the two following loops were independent of eachother:

```
/* d2 as the squared distance between the particles */
for(k=0 ; k < ndim ; k++)
{
    rij[k] = pos[i][k] - pos[j][k];
}

d2 = 0;
for(k=0 ; k < ndim ; k++)
{
    d2 += rij[k]*rij[k];
}
d = sqrt(d2);
```

so they were merged into one loop with no change in results:

```
/* d2 as the squared distance between the particles */
d2 = 0;
for(k=0 ; k < ndim ; k++)
{
    rij[k] = pos[i][k] - pos[j][k];
    d2 += rij[k]*rij[k];
}
d = sqrt(d2);
```

Similarly the loop to calculate the kinetic energy was put into the loop which initialises `rij` since `vel` is not updated anywhere in the function:

```
for(k=0 ; k < ndim ; k++)
{
    /* Initialise forces to zero */
    force[i][k] = 0.0;
    /* compute kinetic energy */
    KE = KE + vel[i][k]*vel[i][k];
}
```

This had no effect on the runtime of either version.

The `dv(d)/d` calculation was being run three times for every inner loop, so this was moved outside the inner loop from:

```
/* Update the force on particle i*/
for(k=0 ; k < ndim ; k++)
{
    force[i][k] = force[i][k] - rij[k]*dv(d)/d;
}
```

To:

```
double dvd = dv(d)/d;
/* Update the force on particle i*/
for(k=0 ; k < ndim ; k++)
{
    force[i][k] = force[i][k] - rij[k]*dvd;
}
```

The unoptimised version now runs in about 30 seconds but there was no change in the runtime of the optimised version.

The value that `k` takes in the inner-most loops of `compute()` is always 3 so these loops were unrolled from:

```
d2 = 0;
for(k=0 ; k < ndim ; k++)
{
    rij[k] = pos[i][k] - pos[j][k];
    d2 += rij[k]*rij[k];
}
d = sqrt(d2);
```

and

```
for(k=0 ; k < ndim ; k++)
{
    force[i][k] = force[i][k] - rij[k]*dvd;
}
```

To:

```
d2 = 0;
rij[0] = pos[i][0] - pos[j][0];
d2 += rij[0]*rij[0];
rij[1] = pos[i][1] - pos[j][1];
d2 += rij[1]*rij[1];
rij[2] = pos[i][2] - pos[j][2];
d2 += rij[2]*rij[2];
d = sqrt(d2);
```

and

```
force[i][0] = force[i][0] - rij[0]*dvd;
force[i][1] = force[i][1] - rij[1]*dvd;
force[i][2] = force[i][2] - rij[2]*dvd;
```

At this point the unoptimised version ran in the following times:

User Time (s)	System Time (s)	Elapsed Time (s)	CPU Usage
27.190	0.078	27.34	99.7%
29.148	0.083	29.29	99.7%
28.075	0.083	28.21	99.7%
28.378	0.078	28.50	99.7%
28.356	0.093	28.51	99.7%

and the optimised version:

User Time (s)	System Time (s)	Elapsed Time (s)	CPU Usage
12.222	0.024	12.26	99.8%
11.908	0.012	11.93	99.8%
12.412	0.028	12.45	99.8%
11.715	0.006	11.72	99.9%
11.640	0.005	11.64	100.0%

The results for both versions were:

PE	KE	Error
49854482.95817039161920547485	0.00000000000000000000	0.00000000000000000000
49854482.95817039161920547485	0.00243672236278765072	0.00000000004887664439

The manual optimisations made no significant difference to the runtime of the optimised version showing that the compiler had already made all those optimisations. It is likely that there was some sort of algorithmic optimisation that could be made to speed the code up significantly but this could not be found.