Optimising Compilers 2004 – Paper 8 Question 7 (AM)

[Syllabus: "strength reduction" and 'available expressions"]
[Thanks to Anton Lokhmotov for corrections to these notes]

(a) The strength reduction optimisation replaces a more expensive operator with a cheaper one; in loops it can remove instructions from the loop body. Given a loop

```
for (i=0; i<M; i++) { k = i*N; ... }
```

it replaces this with a loop

```
k = 0; for (i=0; i<M; (i++; k=k+N)) { ... }
```

The loop can then be transformed on a test using `k`, i.e.

```
for ((i=0,k = 0); k<M*N; (i++, k = k+N)) { ... }
```

and then `i` can be eliminated givin

```
for (k = 0; k<M*N; k = k+N) { ... }
```

Applying this to the given loop (twice)

```
for (i=0; i<M; i++) for (j=0; j<N; j++) t += v[i*N+j];
```

first gives

```
for (k=0; k<M*N; k++) for (j=0; j<N; j++) t += v[k+j];
```

Now note there is the implicit scaling by 4 in the second loop which can be written

```
for (k=0; k<M*N; k++) for (j=0; j<N; j++) { p = &&v + 4*j + 4*k; t += *p; }
```

where &&v is the byte address of `v`. The inner loop can then be written

```
for (q=&&v; j=0; p<&&v + 4*N; (j++, p+=4))
   { t += *(q + 4*k); }
```

and again using `q` instead of `j` using normal pointer notation

```
int *q;
for (q=&v[0]; p<&v[N]; p++)) { t += q[k]; }
```

I.e. we can optimise the original loop to

```
int *q;
for (k=0; k<M*N; k=k+N) for (q=&v[0]; p<&v[N]; p++) { t += q[k]; }
```

This is as far as the techniques presented in lectures provided us to go, although some students may then argue (no extra marks, and a loss if not clearly justified) that we can optimise to

```
for (q=&v; q<&v[M*N]; q++) { t += *q; }
```

using "loop merging".

(b) An expression e is *available* at a program n if it is calculated on every path to node n and not invalidated by any intervening assignment to the free variables of e. Computed "just before entry" means "just before the first iteration, not before the whole loop" in case the expression may cause and exception. Thus in

```
extern int u[100],v[100],w[100];
void f(int n)
{   int i, y = ..., z = ...;
    for (i=5; i<n; i++)
    {    u[i] += 1000/y;
         v[i] += 1000/z;
         p(&y);
         w[i] += 1000/z;
    }
}
```

we have that $1000/y$ and $1000/z$ is available during the loop, but only the latter can be safely assured not have its variables updated by `p`. So we can write the loop first as

```
void f(int n)
{   int i, y = ..., z = ...;
    if (5<n)     /* i.e. repeat the test i<n two lines on from here */
    {   int t = 1000/z;
        for (i=5; i<n; i++)
        {   u[i] += 1000/y;
            v[i] += t;
            p(&y);
            w[i] += t;
         }
    }
}
```