Sheet 07

PS Parallel Programming

Patrick Wintner

May 5, 2025

1 Parallelisizing Loops

The dependencies and parallelisation posibilities of code snippets are analysized.

1.1

1.1.1 Serial

```
for (int i=0; i < n-1; i++) {
        x[i] = (y[i] + x[i+1]) / 7; // S
}
```

Statement S anti-depends (Write-After-Read) on itself: $S\delta^{-1}S$. Anti-dependencies can be eliminated through variable renaming.

1.1.2 Parallel

```
#pragma omp parallel for schedule(static)
for (int i=0; i < n-1; i++) {
        x2[i] = (y[i] + x[i+1]) / 7;
}
```

1.2

1.2.1 Serial

```
for (int i=0; i < n; i++) {
        a = (x[i] + y[i]) / (i+1); // S1
        z[i] = a; // S2
}

f = sqrt(a + k); // S3
```

Statement S2 truly depends (Read-After-Write) on S1 and S3 truly depends on the last instance of S1: $S1\delta S2, S2\delta S3$. The depency is obviously not loop-carried, therefore the loop can be parallelized by making 'a' private within the loop.

1.2.2 Parallel

```
#pragma omp parallel for lastprivate(a)
for (int i=0; i < n; i++) {
        a = (x[i] + y[i]) / (i+1);
        z[i] = a;
}

f = sqrt(a + k);
```

1.3

1.3.1 Serial

```
for (int i=0; i < n; i++) {
        x[i] = y[i] * 2 + b * i; // S1
}

for (int i=0; i < n; i++) {
        y[i] = x[i] + a / (i+1); // S2
}
```

Statement S2 both truly and anti-depends on S1: $S1\delta S2, S1\delta^{-1}S2$. There is no dependence within the loops, therefore the loops themselves can be parallelized.

1.3.2 Parallel

```
#pragma omp parallel for schedule(static)
for (int i=0; i < n; i++) {
        x[i] = y[i] * 2 + b * i;
}

#pragma omp parallel for schedule(static)
for (int i=0; i < n; i++) {
        y[i] = x[i] + a / (i+1);
}
```

2 Parallelisizing more Loops

The dependencies of code snippets are analysized and the code snippets themselves are parallelized. The wall time of both the serial and parallel versions is measured.

2.1

2.1.1 Serial

```
#ifndef A_SER_H
#define A_SER_H

double factor = 1;

for (int i=0; i < n; i++) {
        x[i] = factor * y[i];
        factor = factor / 2;
}
#endif
```

2.1.2 Parallel

```
#ifndef A_PAR_H
#define A_PAR_H

double factor = 1;

for (int i=0; i < n; i++) {
        x[i] = factor * y[i];
        factor = factor / 2;
}
#endif
```

2.1.3 Serial

```
for (int i=1; i<n; i++) {
        x[i] =(x[i] + y[i-1]) / 2;
        y[i] = y[i] +z[i] * 3;
}
```

2.1.4 Parallel

```
for (int i=1; i<n; i++) {
        x[i] =(x[i] + y[i-1]) / 2;
        y[i] = y[i] +z[i] * 3;
}
```

2.1.5 Serial

```
x[0] = x[0] + 5 * y[0];
for (int i = 1; i<n; i++) {
        x[i] = x[i] + 5 * y[i];
        if ( twice ) {
                x[i-1] = 2 * x[i-1];
        }
}
```

2.1.6 Parallel

```
x[0] = x[0] + 5 * y[0];
for (int i = 1; i<n; i++) {
        x[i] = x[i] + 5 * y[i];
        if ( twice ) {
                x[i-1] = 2 * x[i-1];
        }
}
```