A summary of the equations for instruction scheduling

NX = 60, NY = 60

Unroll by 3 (PE=40):

No. of operations = 18*(NX/3)*(NY/3)*2 = 18*20*20*2 = 14400

No. of iterations outside innermost loop (i and j) = 2*(NX/(3*PE/2))*(NY/3) = 40

No. of cycles = 18*20 = 14400/40 = 360

Unroll by 4 (PE=30):

No. of operations = 24*(NX/4)*(NY/3)*2 = 24*15*20*2 = 14400

No. of iterations outside innermost loop (i and j) = 2*(NX/(4*PE/2))*(NY/3) = 40

No. of cycles = 24*20 = 14400/30 = 480

Unroll by 5 (PE=24):

No. of operations = 30*(NX/5)*(NY/3)*2 = 30*12*20*2 = 14400

No. of iterations outside innermost loop (i and j) = 2*(NX/(5*PE/2))*(NY/3) = 40

No. of cycles = 30*20 = 14400/24 = 600

To determine the **minimum No. of PEs** for tmp (PE1) and y (PE2), we could use the following criterion under certain constraint (innermost loop unroll factor = 4):

Min {Max [(NX/(4*PE1))*(NY/3) + (NX/3)((NY/(4*PE2))]}

Constraint: PE = (PE1+PE2) <= Max No. of PE

To have a better understanding of the scheduling for *atax* kernel (NX = 60, NY = 60), I have attached the source code and the code after loop unrolling (unrolling factor = 3 or 4 or 5) and loop tiling (tiling factor = 20 or 15 or 12).

In this specific case, the internal computation path has 3 or 4 or 5 pipeline stages, and the total No. of PE is 40 (PE1 = PE2 = 20) for 3-stage design, or 30 (PE1 = PE2 = 15) for 4-stage pipeline, or 24 (PE1 = PE2 = 12) for 5-stage pipeline.

```
#define NX 60
#define NY 60
int i;
int j;
int tmp[NX];
int y[NY];
int x[NY];
int A[NX][NY];

int main()
{
  for (i = 0; i < NY; i += 1) {
    y[i] = 0;
  }
  for (i = 0; i < NX; i += 1) {
    tmp[i] = 0;
  }
  for (i = 0; i < NX; i += 1) {
    for (j = 0; j < NY; j += 1) {
      tmp[i] = tmp[i] + A[i][j] * x[j];
    }
  }
  for (i = 0; i < NX; i += 1) {
    for (j = 0; j < NY; j += 1) {
      y[j] = y[j] + A[i][j] * tmp[i];
    }
  }
}
```

```
#define NX 60
#define NY 60
int i;
int j;
int tmp[60];
int y[60];
int x[60];
int A[60][60];

int main()
{
  for (i = 0; i < 60; i += 1) {
    y[i] = 0;
  }
  for (i = 0; i < 60; i += 1) {
    tmp[i] = 0;
  }
  int ii;
  for (ii = 0; ii <= 59; ii += 60) {
    for (i = ii; i <= ((59 < ii + 20 - 1?59 : ii + 20*3 - 1)); i += 3) {
      for (j = 0; j <= 59; j += 3) {
        tmp[i] = tmp[i] + A[i][j] * x[j] + A[i][j + 1] * x[j + 1] + A[i][j + 2] * x[j + 2];
        tmp[i + 1] = tmp[i + 1] + A[i + 1][j] * x[j] + A[i + 1][j + 1] * x[j + 1] + A[i + 1][j + 2] * x[j + 2];
        tmp[i + 2] = tmp[i + 2] + A[i + 2][j] * x[j] + A[i + 2][j + 1] * x[j + 1] + A[i + 2][j + 2] * x[j + 2];
      }
    }
  }
  int jj;
  for (jj = 0; jj <= 59; jj += 60) {
    for (j = jj; j <= ((59 < jj + 20 - 1?59 : jj + 20*3 - 1)); j += 3) {
      for (i = 0; i <= 59; i += 3) {
        y[j] = y[j] + A[i][j] * tmp[i] + A[i + 1][j] * tmp[i + 1] + A[i + 2][j] * tmp[i + 2];
        y[j + 1] = y[j + 1] + A[i][j + 1] * tmp[i] + A[i + 1][j + 1] * tmp[i + 1] + A[i + 2][j + 1] * tmp[i + 2];
        y[j + 2] = y[j + 2] + A[i][j + 2] * tmp[i] + A[i + 1][j + 2] * tmp[i + 1] + A[i + 2][j + 2] * tmp[i + 2];
      }
    }
  }
}
```

```
#define NX 60
#define NY 60
int i;
int j;
int tmp[60];
int y[60];
int x[60];
int A[60][60];

int main()
{
  for (i = 0; i < 60; i += 1) {
    y[i] = 0;
  }
  for (i = 0; i < 60; i += 1) {
    tmp[i] = 0;
  }
  int ii;
  for (ii = 0; ii <= 59; ii += 60) {
    for (i = ii; i <= ((59 < ii + 15*4 - 1?59 : ii + 15*4 - 1)); i += 4) {
      for (j = 0; j <= 59; j += 3) {
        tmp[i] = tmp[i] + A[i][j] * x[j] + A[i][j + 1] * x[j + 1] + A[i][j + 2] * x[j + 2];
        tmp[i + 1] = tmp[i + 1] + A[i + 1][j] * x[j] + A[i + 1][j + 1] * x[j + 1] + A[i + 1][j + 2] * x[j + 2];
        tmp[i + 2] = tmp[i + 2] + A[i + 2][j] * x[j] + A[i + 2][j + 1] * x[j + 1] + A[i + 2][j + 2] * x[j + 2];
        tmp[i + 3] = tmp[i + 3] + A[i + 3][j] * x[j] + A[i + 3][j + 1] * x[j + 1] + A[i + 3][j + 2] * x[j + 2];
      }
    }
  }
  int jj;
  for (jj = 0; jj <= 59; jj += 60) {
    for (j = jj; j <= ((59 < jj + 15*4 - 1?59 : jj + 15*4 - 1)); j += 4) {
      for (i = 0; i <= 59; i += 3) {
        y[j] = y[j] + A[i][j] * tmp[i] + A[i + 1][j] * tmp[i + 1] + A[i + 2][j] * tmp[i + 2];
        y[j + 1] = y[j + 1] + A[i][j + 1] * tmp[i] + A[i + 1][j + 1] * tmp[i + 1] + A[i + 2][j + 1] * tmp[i + 2];
        y[j + 2] = y[j + 2] + A[i][j + 2] * tmp[i] + A[i + 1][j + 2] * tmp[i + 1] + A[i + 2][j + 2] * tmp[i + 2];
        y[j + 3] = y[j + 3] + A[i][j + 3] * tmp[i] + A[i + 1][j + 3] * tmp[i + 1] + A[i + 2][j + 3] * tmp[i + 2];
      }
    }
  }
}
```

```
#define NX 60
#define NY 60
int i;
int j;
int tmp[60];
int y[60];
int x[60];
int A[60][60];

int main()
{
  for (i = 0; i < 60; i += 1) {
    y[i] = 0;
  }
  for (i = 0; i < 60; i += 1) {
    tmp[i] = 0;
  }
  int ii;
  for (ii = 0; ii <= 59; ii += 60) {
    for (i = ii; i <= ((59 < ii + 12*5 - 1?59 : ii + 12*5 - 1)); i += 5) {
      for (j = 0; j <= 59; j += 3) {
        tmp[i] = tmp[i] + A[i][j] * x[j] + A[i][j + 1] * x[j + 1] + A[i][j + 2] * x[j + 2];
        tmp[i + 1] = tmp[i + 1] + A[i + 1][j] * x[j] + A[i + 1][j + 1] * x[j + 1] + A[i + 1][j + 2] * x[j + 2];
        tmp[i + 2] = tmp[i + 2] + A[i + 2][j] * x[j] + A[i + 2][j + 1] * x[j + 1] + A[i + 2][j + 2] * x[j + 2];
        tmp[i + 3] = tmp[i + 3] + A[i + 3][j] * x[j] + A[i + 3][j + 1] * x[j + 1] + A[i + 3][j + 2] * x[j + 2];
        tmp[i + 4] = tmp[i + 4] + A[i + 4][j] * x[j] + A[i + 4][j + 1] * x[j + 1] + A[i + 4][j + 2] * x[j + 2];
      }
    }
  }
  int jj;
  for (jj = 0; jj <= 59; jj += 60) {
    for (j = jj; j <= ((59 < jj + 12*5 - 1?59 : jj + 12*5 - 1)); j += 5) {
      for (i = 0; i <= 59; i += 3) {
        y[j] = y[j] + A[i][j] * tmp[i] + A[i + 1][j] * tmp[i + 1] + A[i + 2][j] * tmp[i + 2];
        y[j + 1] = y[j + 1] + A[i][j + 1] * tmp[i] + A[i + 1][j + 1] * tmp[i + 1] + A[i + 2][j + 1] * tmp[i + 2];
        y[j + 2] = y[j + 2] + A[i][j + 2] * tmp[i] + A[i + 1][j + 2] * tmp[i + 1] + A[i + 2][j + 2] * tmp[i + 2];
        y[j + 3] = y[j + 3] + A[i][j + 3] * tmp[i] + A[i + 1][j + 3] * tmp[i + 1] + A[i + 2][j + 3] * tmp[i + 2];
        y[j + 4] = y[j + 4] + A[i][j + 4] * tmp[i] + A[i + 1][j + 4] * tmp[i + 1] + A[i + 2][j + 4] * tmp[i + 2];
      }
    }
  }
}
```