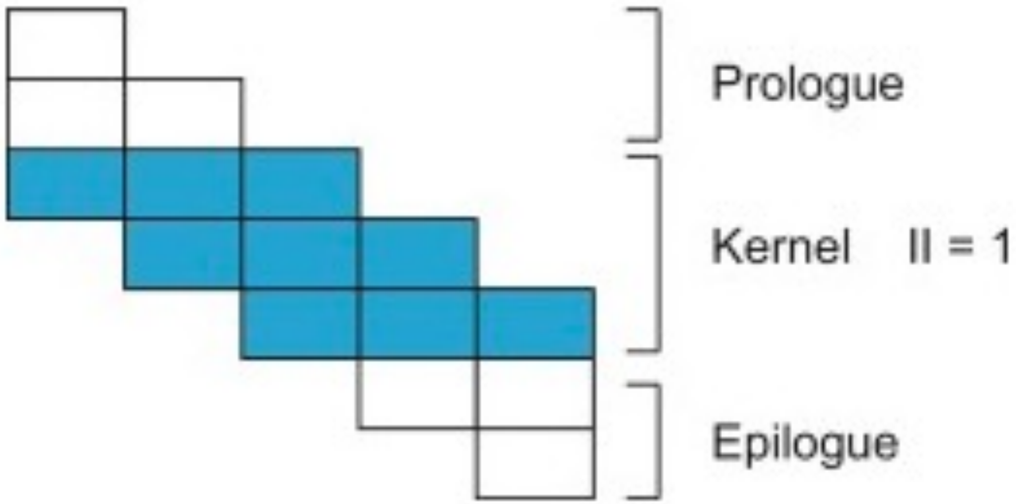
CS4473/5473: Parallel, Distributed, and Network Programming (PDN)

Dr. Richard Veras

Week 11 Lecture 15



Assignments Coming Down the Pipe

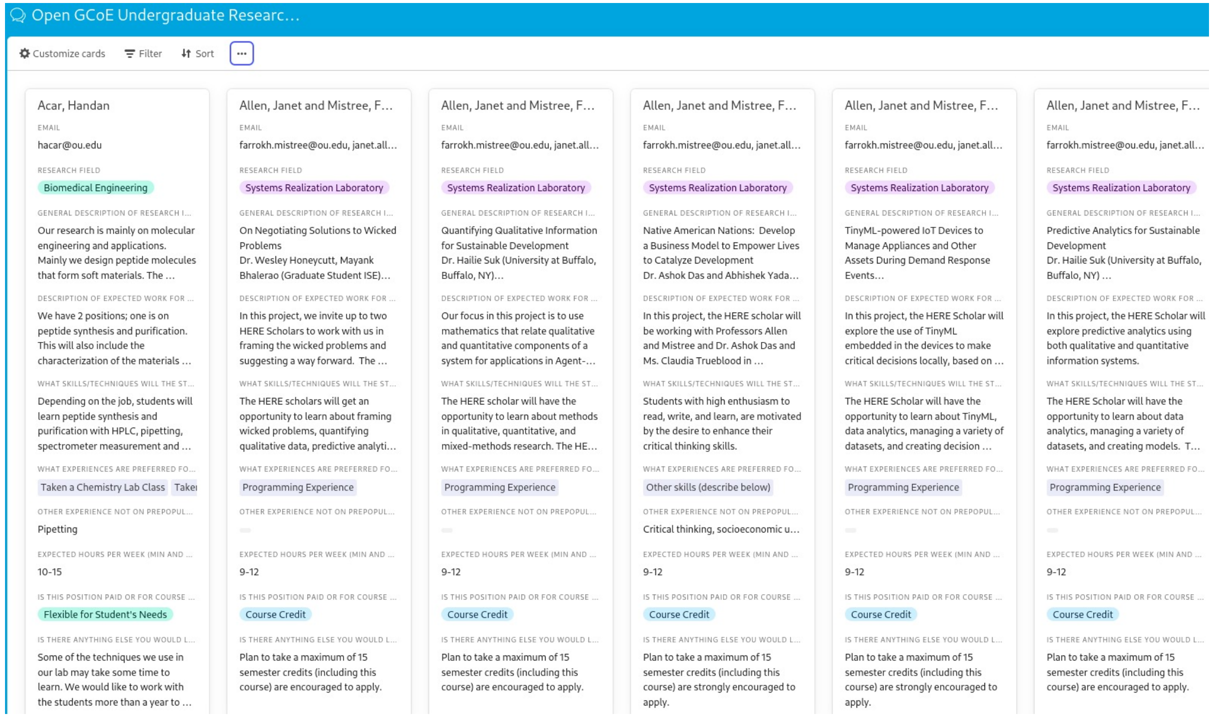
- Reading Assignment:
  - Ch 3 of An introduction to Parallel Programming 2e
- Programming Assignments:
  - C Refresher part II
  - C and Irregular Data
- Labs
  - When Bubble Sort Beats Quick Sort
- 5473 Readings & Reviews
  - Software Pipelining
  - X-Ray

Undergraduate Research Opportunities

Interest in Project Form: Due Friday at 5pm:
https://airtable.com/shrfzgUwgKWcz0E70

Listing of Projects:
https://airtable.com/shryOJoG5uVZRerby

Find projects/labs you are interested in and fill out the form to



Undergraduate Research Opportunities

Once you decide on a lab and project, work with the professor to fill out this form to be in the research class 3980 (This can count towards CS Elective hours):
https://www.ou.edu/honors/honors-forms

Honors Research

Honors Research is an independent study course required to graduate with Honors, taken by juniors and seniors with the professor of their choice to mentor them in conducting their own research or assisting the professor with their research. This is intended to be your opportunity to research the topic of your choice and gain practical experience in your field. Research does not necessarily have to be taken in your major department, but you must check with your department to see if there are any requirements or restrictions. This is your opportunity to produce your Honors Thesis, which is due to the Honors College at the end of the semester you graduate. A total of 3 credit hours of Research is required of all majors. Below is the full Honors College policy on Research and Thesis requirements. **Your Honors Research enrollment form is due to the Honors College main office with all signatures but the Honors Dean by the end of the add/drop period.**

Honors Research (INSTRUCTIONS)
Honors Research Form

If you are not in the honors program, don't panic! You can petition to get into an honors class (CS 3980, ECE 3980, etc). If you do not meet the requirements and want to do research, still apply because the Professor gives the permission.

Petition to take an Honors Course as a Non-Honors Student

Petition forms are for non-honors students who wish to take an honors course. You may be issued permission to enroll if:

-The course is not fully enrolled.
-All active Honors students have had the chance to enroll in the course. This typically occurs the week after freshmen have enrolled in mid-April or mid-November.
-The student has maintained a 3.40 OU retention GPA or above.

OR

-The student has received the instructor's permission to enroll in the course. We will need proof of this permission, which can be attached to this form or emailed to Tanya Miller at maverickly@ou.edu.

Petition to Enroll in an Honors-Designated Course as a Non-Honors Student

Outline

- Administrivia
- Recap
- Loops

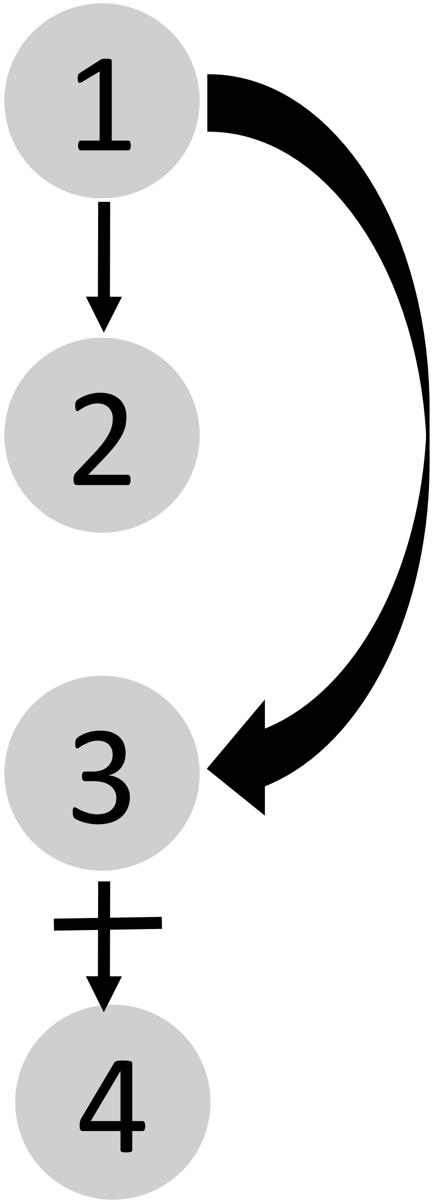
Loop Transformations

- Dependency Analysis
- Loop Peeling
- Loop Fusion
- Loop Fission
- Loop Interchanging
- Loop Skewing
- Strip-Mining
- Loop Tiling

Data Dependence

Can we reorder this code?

```
(1) A = 0
(2) B = A
(3) C = A + D
(4) D = 2
```



"High Performance Compilers for Parallel Computing." Wolfe, Michael.

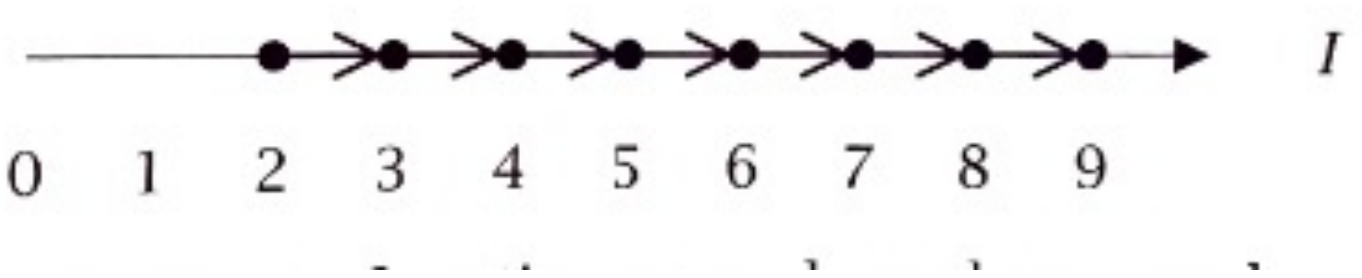
Data Dependence and Loops

Have loop with dependencies within and between iterations.

```
(1)   for I = 2 to 9 do
(2)       X[I] = Y[I] + Z[I]
(3)       A[I] = X[I-1] + 1
(4)   endfor
```

	I = 2	I = 3	I = 4
(2)	X[2]=Y[2]+Z[2]	X[3]=Y[3]+Z[3]	X[4]=Y[4]+Z[4]
(3)	A[2]=X[1]+1	A[3]=X[2]+1	A[4]=X[3]+1

We can view the dependencies across the iteration space.



"High Performance Compilers for Parallel Computing." Wolfe, Michael.

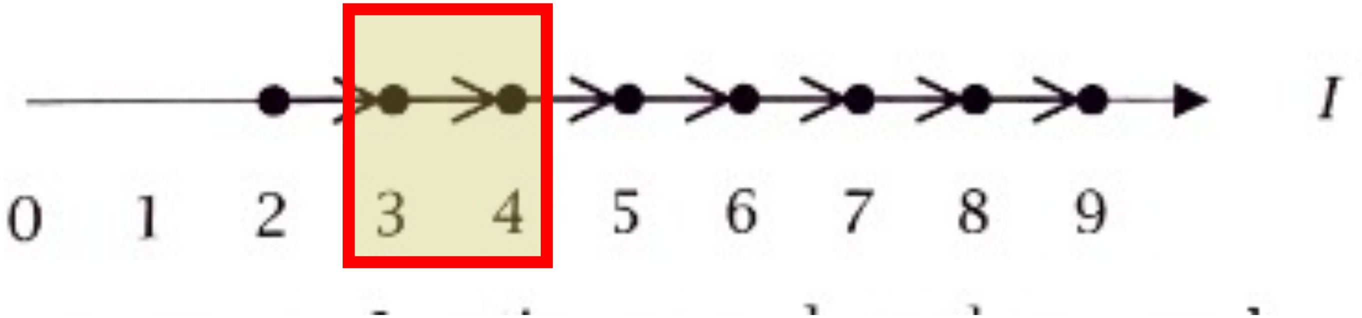
Data Dependence and Loops

Have loop with dependencies within and between iterations.

	I = 2	I = 3	I = 4
(2)	X[2]=Y[2]+Z[2]	X[3]=Y[3]+Z[3]	X[4]=Y[4]+Z[4]
(3)	A[2]=X[1]+1	A[3]=X[2]+1	A[4]=X[3]+1

```
(1)   for I = 2 to 9 do
(2)       X[I] = Y[I] + Z[I]
(3)       A[I] = X[I-1] + 1
(4)   endfor
```

We can view the dependencies across the iteration space.



"High Performance Compilers for Parallel Computing." Wolfe, Michael.

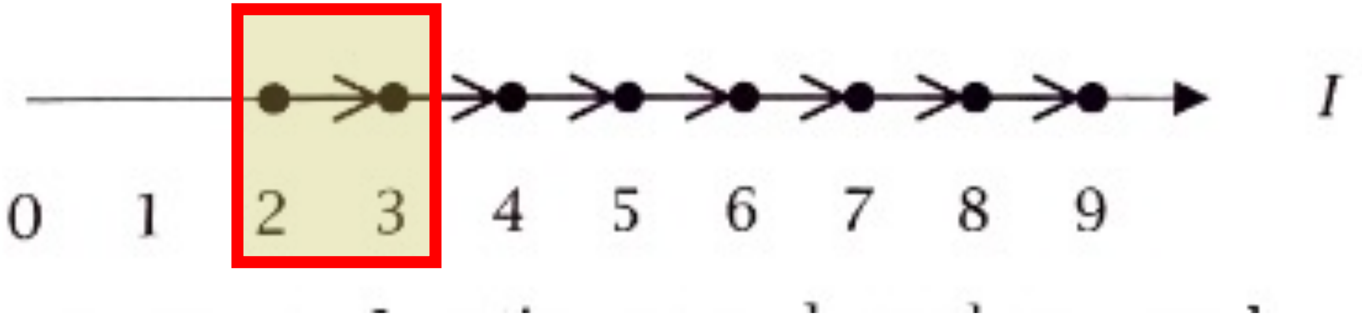
Data Dependence and Loops

Have loop with dependencies within and between iterations.

	I = 2	I = 3	I = 4
(2)	X[2]=Y[2]+Z[2]	X[3]=Y[3]+Z[3]	X[4]=Y[4]+Z[4]
(3)	A[2]=X[1]+1	A[3]=X[2]+1	A[4]=X[3]+1

```
(1)   for I = 2 to 9 do
(2)       X[I] = Y[I] + Z[I]
(3)       A[I] = X[I-1] + 1
(4)   endfor
```

We can view the dependencies across the iteration space.



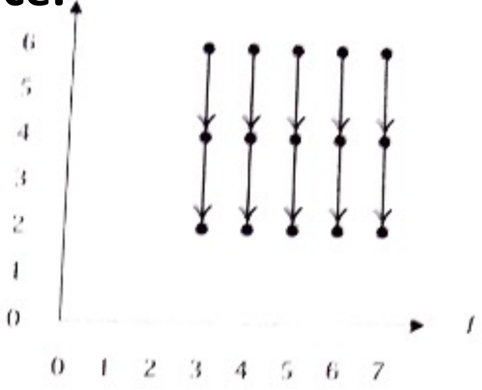
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Iteration Space for Nested Loops.

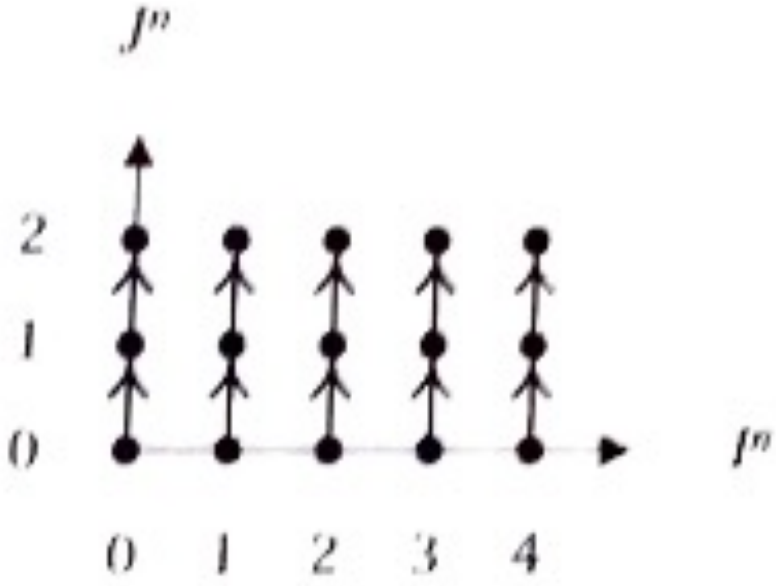
Have nested loops with dependencies between iterations.

```
for I = 3 to 7 do
    for J = 6 to 2 by -2 do
        A[I,J] = A[I,J+2] + 1
    endfor
endfor
```

This gives us a multidimensional iteration space.



That can be normalized ("cleaned up").



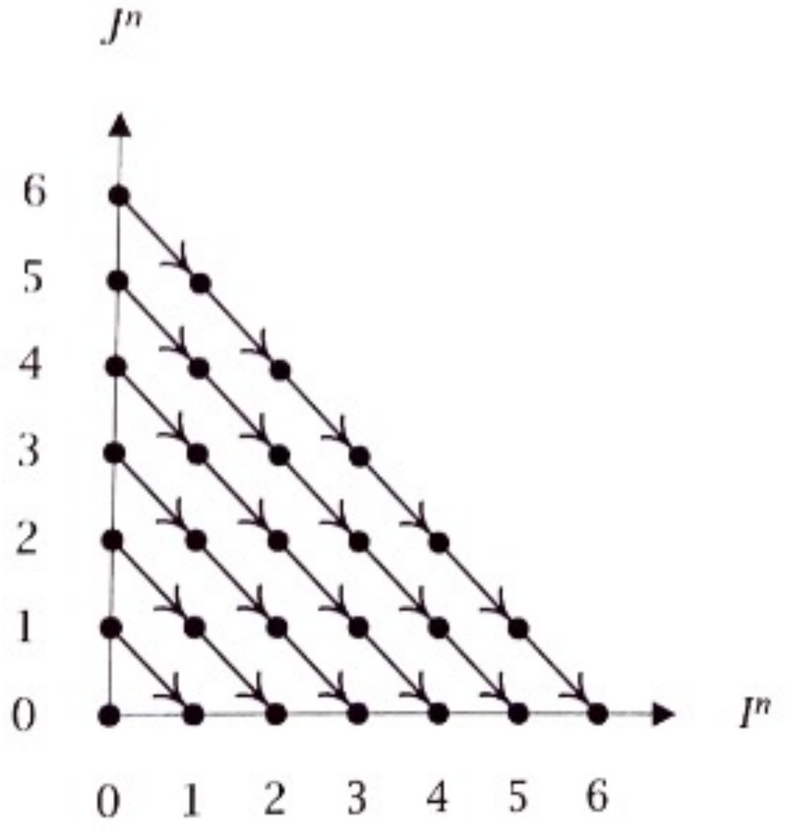
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
for I = 1 to 7 do
    for J = I to 7 do
        A[I+1,J] = A[I,J] + 1
    endfor
endfor
```

Iteration space for the loops.



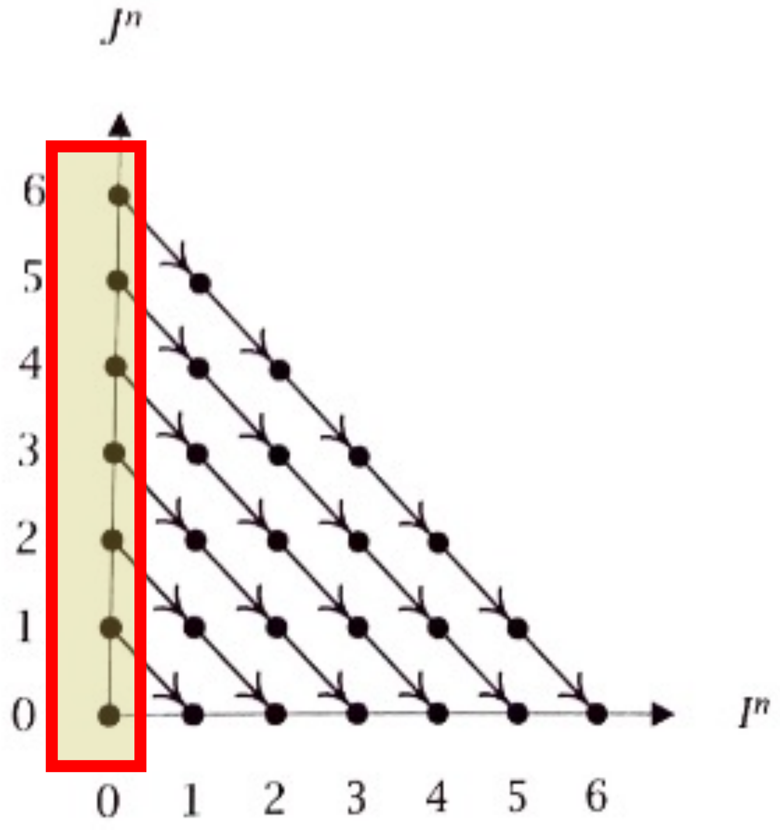
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)   for I = 1 to 7 do
(2)       for J = I to 7 do
(3)           A[I+1,J] = A[I,J] + 1
(4)       endfor
(5)   endfor
```

Iteration space for the loops.



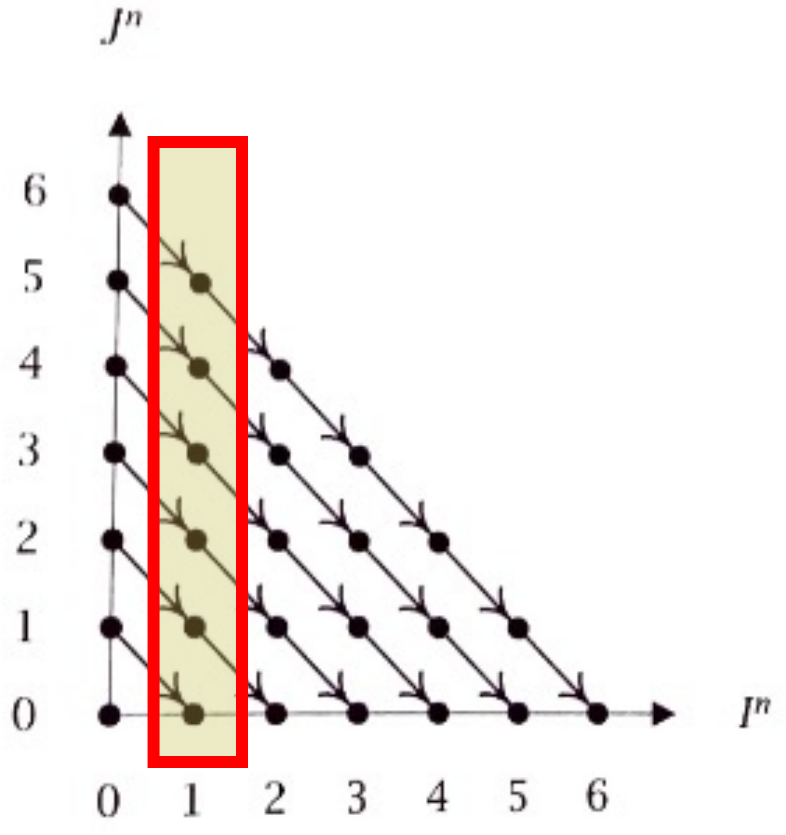
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)  for I = 1 to 7 do
(2)      for J = I to 7 do
(3)          A[I+1,J] = A[I,J] + 1
(4)      endfor
(5)  endfor
```

Iteration space for the loops.



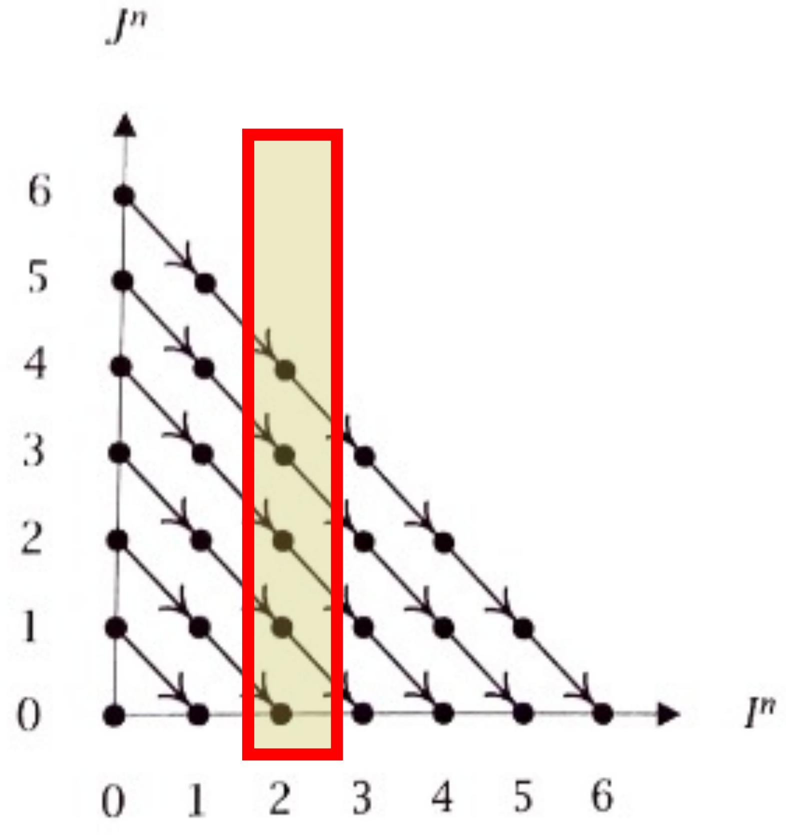
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)   for I = 1 to 7 do
(2)       for J = I to 7 do
(3)           A[I+1,J] = A[I,J] + 1
(4)       endfor
(5)   endfor
```

Iteration space for the loops.



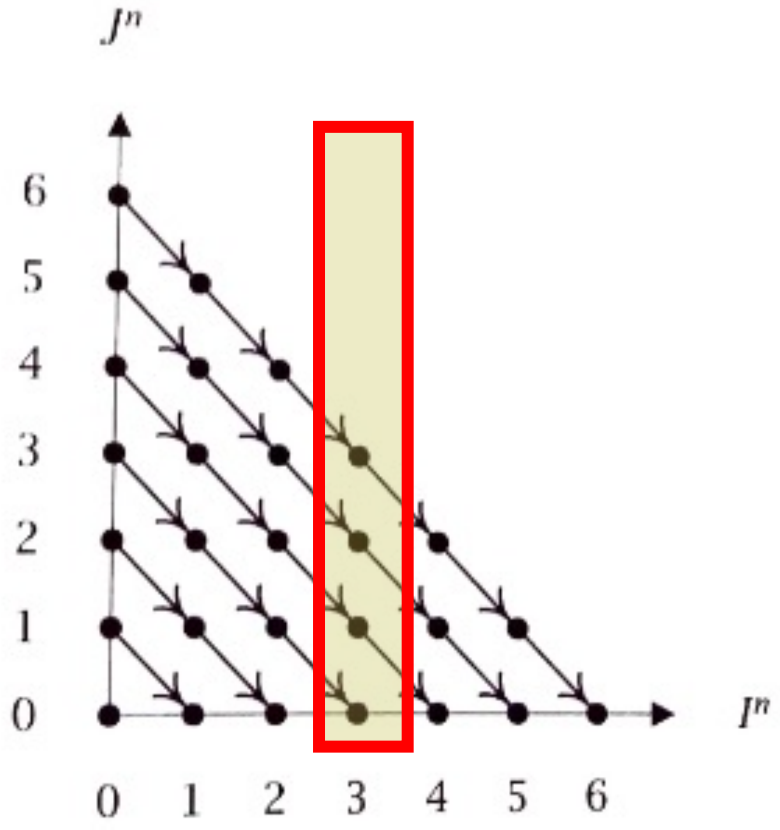
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)   for I = 1 to 7 do
(2)       for J = I to 7 do
(3)           A[I+1,J] = A[I,J] + 1
(4)       endfor
(5)   endfor
```

Iteration space for the loops.



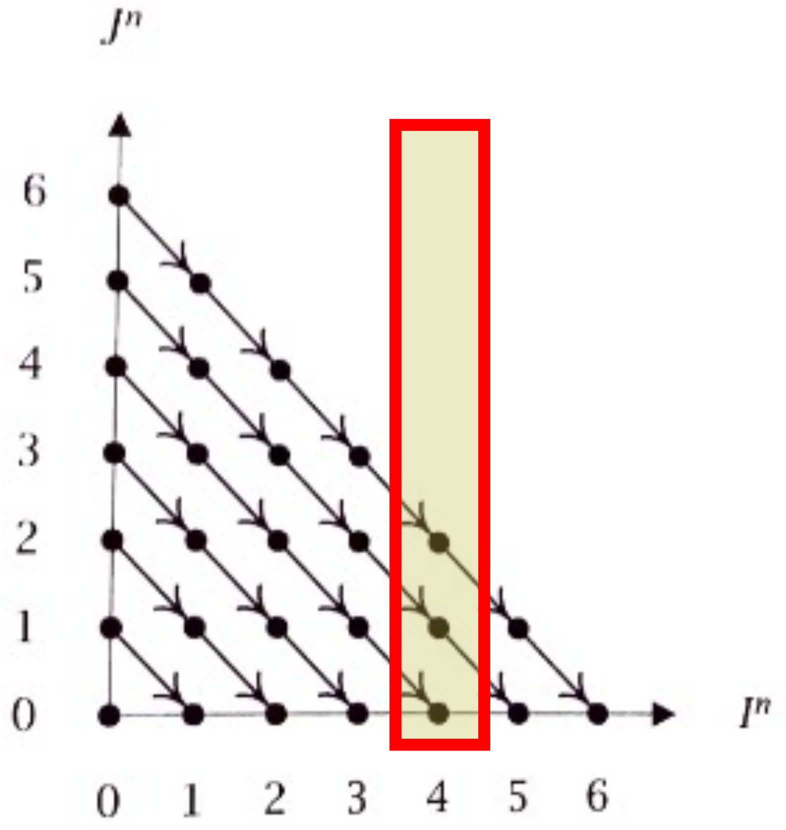
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
for I = 1 to 7 do
    for J = I to 7 do
        A[I+1,J] = A[I,J] + 1
    endfor
endfor
```

Iteration space for the loops.



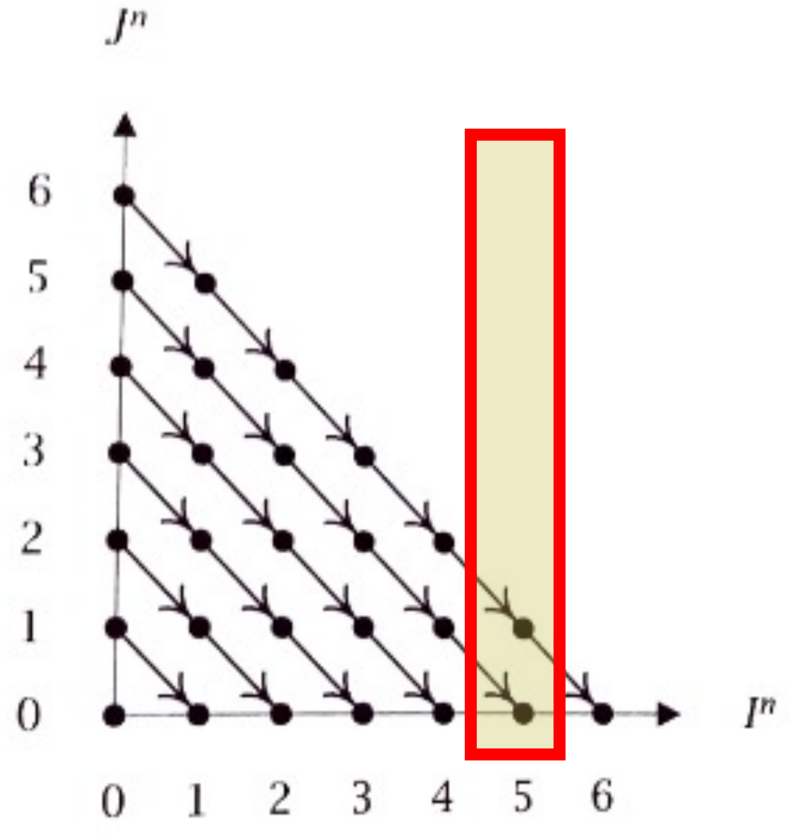
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)    for I = 1 to 7 do
(2)        for J = I to 7 do
(3)            A[I+1,J] = A[I,J] + 1
(4)        endfor
(5)    endfor
```

Iteration space for the loops.



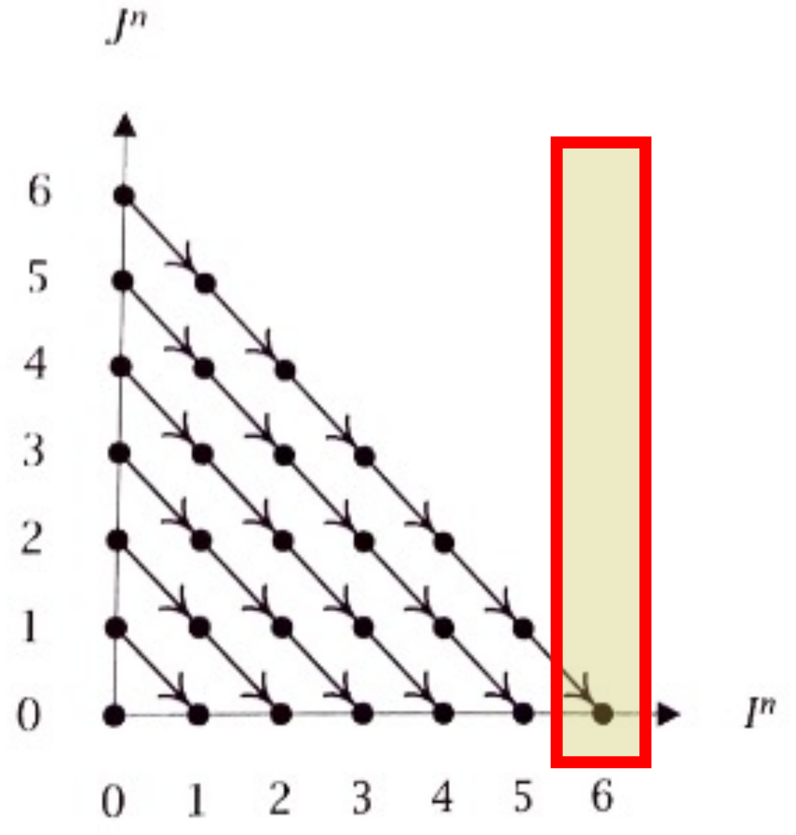
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)   for I = 1 to 7 do
(2)       for J = I to 7 do
(3)           A[I+1,J] = A[I,J] + 1
(4)       endfor
(5)   endfor
```

Iteration space for the loops.



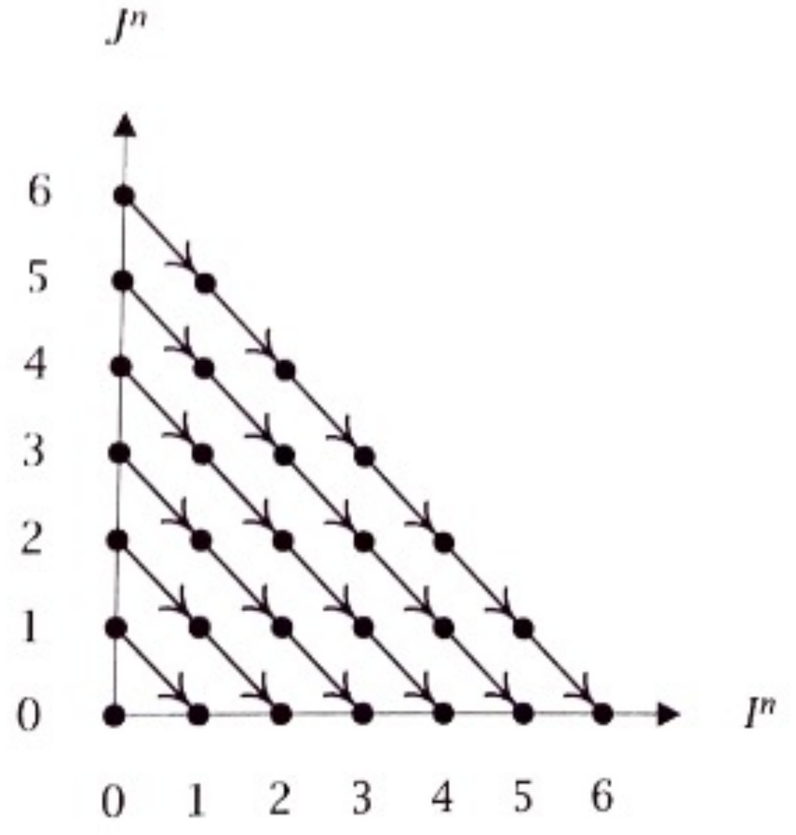
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)   for I = 1 to 7 do
(2)       for J = I to 7 do
(3)           A[I+1,J] = A[I,J] + 1
(4)       endfor
(5)   endfor
```

Iteration space for the loops.



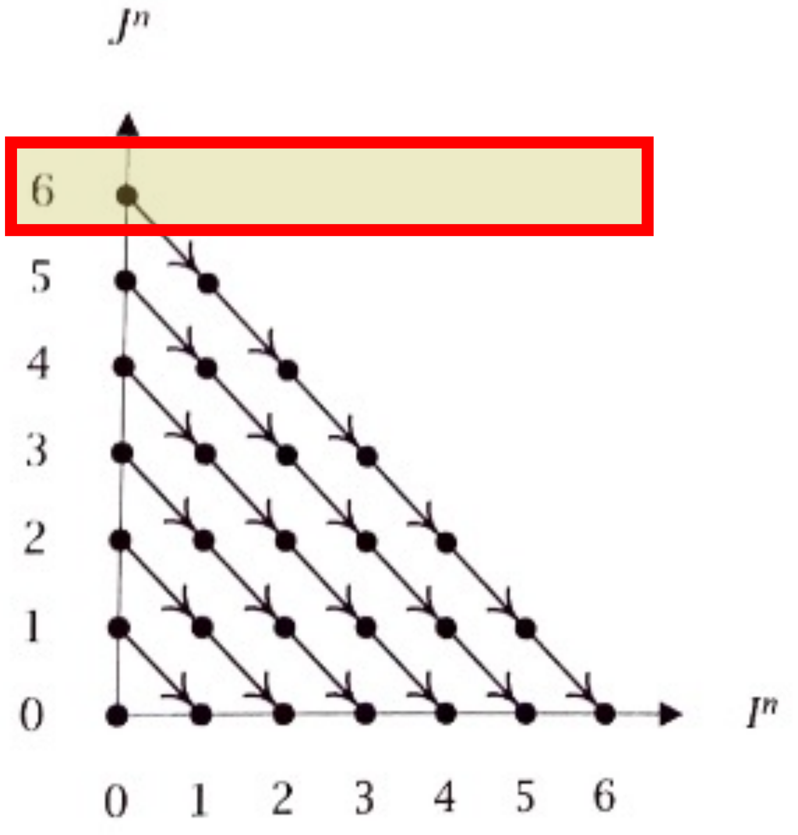
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)    for I = 1 to 7 do
(2)        for J = I to 7 do
(3)            A[I+1,J] = A[I,J] + 1
(4)        endfor
(5)    endfor
```

Iteration space for the loops.



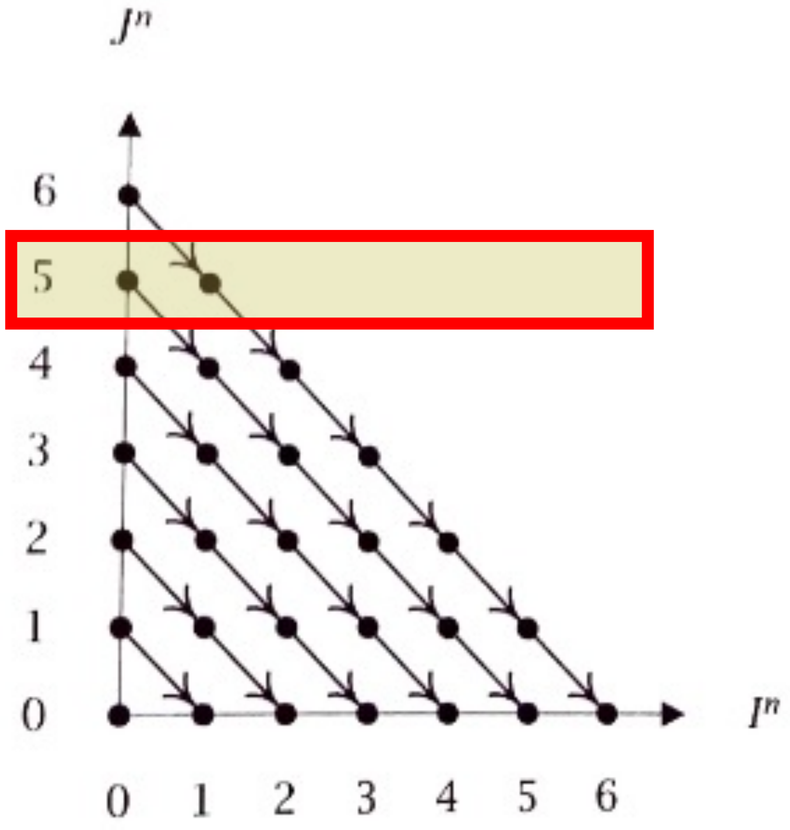
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)  for I = 1 to 7 do
(2)      for J = I to 7 do
(3)          A[I+1,J] = A[I,J] + 1
(4)      endfor
(5)  endfor
```

Iteration space for the loops.



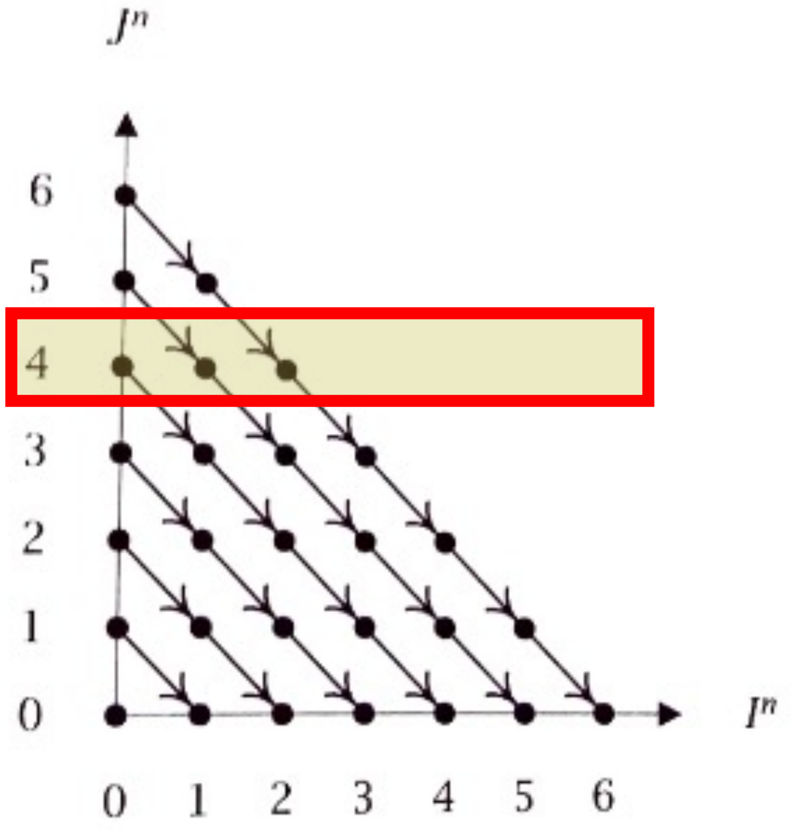
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
for I = 1 to 7 do
    for J = I to 7 do
        A[I+1,J] = A[I,J] + 1
    endfor
endfor
```

Iteration space for the loops.



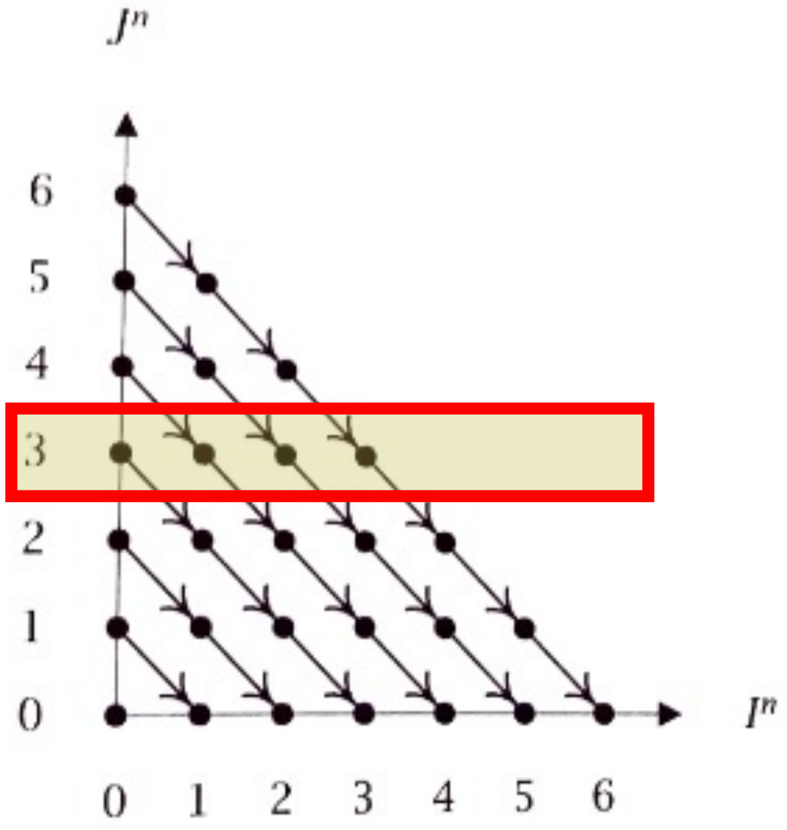
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)  for I = 1 to 7 do
(2)      for J = I to 7 do
(3)          A[I+1,J] = A[I,J] + 1
(4)      endfor
(5)  endfor
```

Iteration space for the loops.



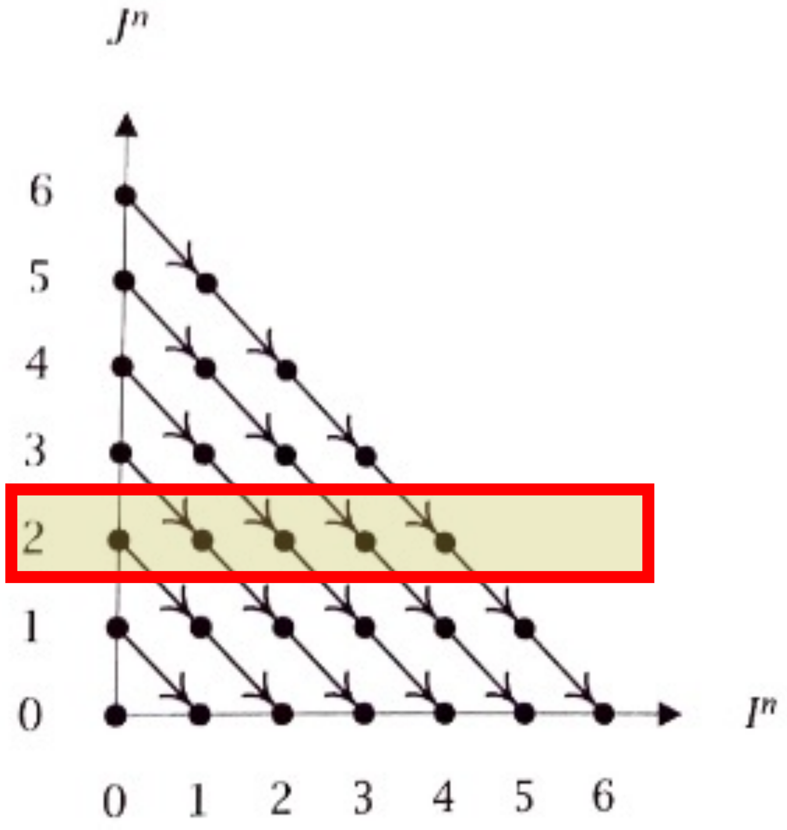
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)   for I = 1 to 7 do
(2)      for J = I to 7 do
(3)         A[I+1,J] = A[I,J] + 1
(4)      endfor
(5)   endfor
```

Iteration space for the loops.



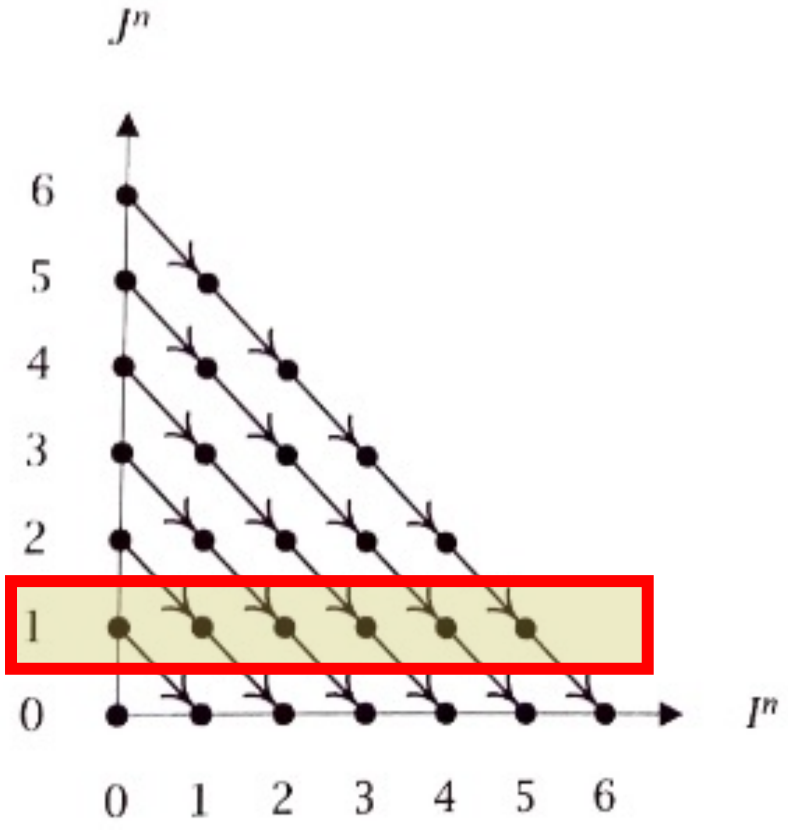
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)  for I = 1 to 7 do
(2)      for J = I to 7 do
(3)          A[I+1,J] = A[I,J] + 1
(4)      endfor
(5)  endfor
```

Iteration space for the loops.



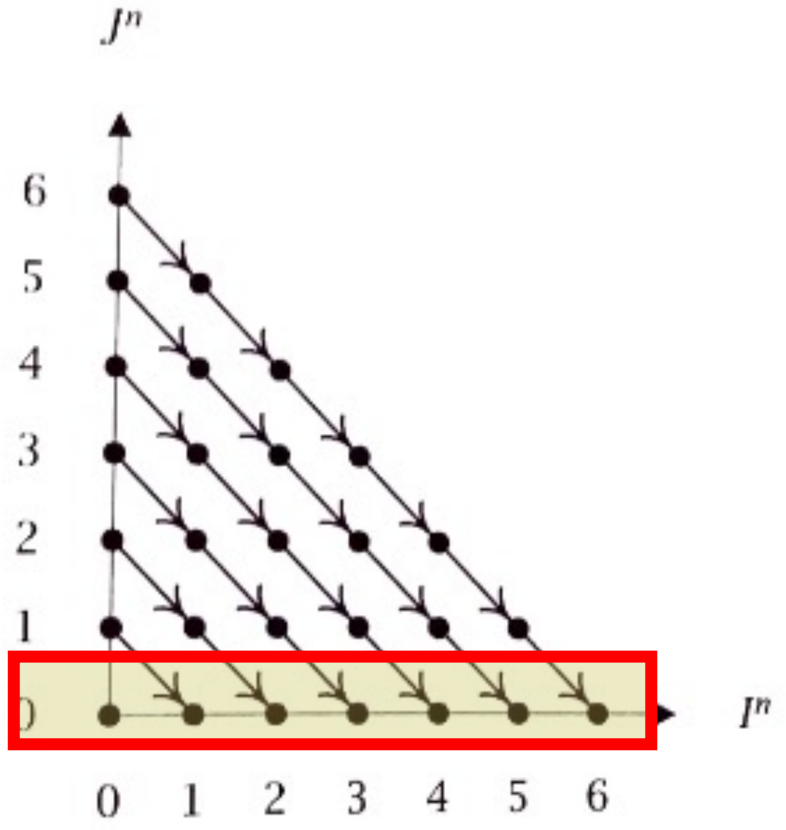
"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Complex Iteration Spaces

Have nested loops with dependencies between iterations.

```
(1)  for I = 1 to 7 do
(2)      for J = I to 7 do
(3)          A[I+1,J] = A[I,J] + 1
(4)      endfor
(5)  endfor
```

Iteration space for the loops.



"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Loop Transformations

- Dependency Analysis
- Simple Transformations
- Loop Peeling
- Loop Fusion
- Loop Fission
- Loop Interchanging
- Loop Skewing
- Strip-Mining
- Loop Tiling

Unswitching

Original

```
(1)   for I = 1 to N do
(2)      for J = 2 to N do
(3)         if T[I] > 0 then
(4)            A[I,J] = A[I,J-1]*T[I] + B[J]
(5)         else
(6)            A[I,J] = 0.0
(7)         endif
(8)      endfor
(9)   endfor
```

Transformed

```
(1)   for I = 1 to N do
(3)      if T[I] > 0 then
(2)         for J = 2 to N do
(4)            A[I,J] = A[I,J-1]*T[I] + B[J]
(8)         endfor
(5)      else
(2)         for J = 2 to N do
(6)            A[I,J] = 0.0
(8)         endfor
(7)      endif
(9)   endfor
```

"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Loop Peeling (Splitting)

Original

```
int p = 10;
for (int i=0; i<10; ++i)
{
  y[i] = x[i] + x[p];
  p = i;
}
```

Transformed

```
y[0] = x[0] + x[10];
for (int i=1; i<10; ++i)
{
  y[i] = x[i] + x[i-1];
}
```

https://en.wikipedia.org/wiki/Loop_splitting

Index Set Splitting

Original

```
(1)    compute tc
(2)    for i = 0 to tc-1 do
(3)        body
(4)    endfor
```

Transformed

```
(1)    compute tc
(2)    for i = 0 to s-1 do
(3)        body
(4)    endfor
(2)    for i = s to tc-1 do
(3)        body
(4)    endfor
```

"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Index Set Splitting

Original

```
(1)   compute tc
(2)   for i = 0 to tc-1 do
(3)       body
(4)   endfor
```

Transformed

```
(1)   compute tc
(2)   for i = 0 to s-1 do
(3)       body
(4)   endfor
(2)   for i = s to tc-1 do
(3)       body
(4)   endfor
```

"High Performance Compilers for Parallel Computing." Wolfe, Michael.

Loop Fusion

Original

```
for (i = 0; i < 300; i++)
  a[i] = a[i] + 3;

for (i = 0; i < 300; i++)
  b[i] = b[i] + 4;
```

Transformed

```
for (i = 0; i < 300; i++)
  {
    a[i] = a[i] + 3;
    b[i] = b[i] + 4;
  }
```

https://compileroptimizations.com/category/loop_fusion.htm

Loop Fusion

Original

```
for (i = 0; i < 300; i++)
  a[i] = a[i] + 3;

for (i = 0; i < 300; i++)
  b[i] = b[i] + 4;
```

Transformed

```
for (i = 0; i < 300; i++)
  {
    a[i] = a[i] + 3;
    b[i] = b[i] + 4;
  }
```

https://compileroptimizations.com/category/loop_fusion.htm

Loop Fission

Original

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
    b[i] = 2;
}
```

Transformed

```
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
}
for (i = 0; i < 100; i++) {
    b[i] = 2;
}
```

https://en.wikipedia.org/wiki/Loop_fission_and_fusion

Loop Fission

Original

```c
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
    b[i] = 2;
}
```

Transformed

```c
int i, a[100], b[100];
for (i = 0; i < 100; i++) {
    a[i] = 1;
}
for (i = 0; i < 100; i++) {
    b[i] = 2;
}
```

https://en.wikipedia.org/wiki/Loop_fission_and_fusion

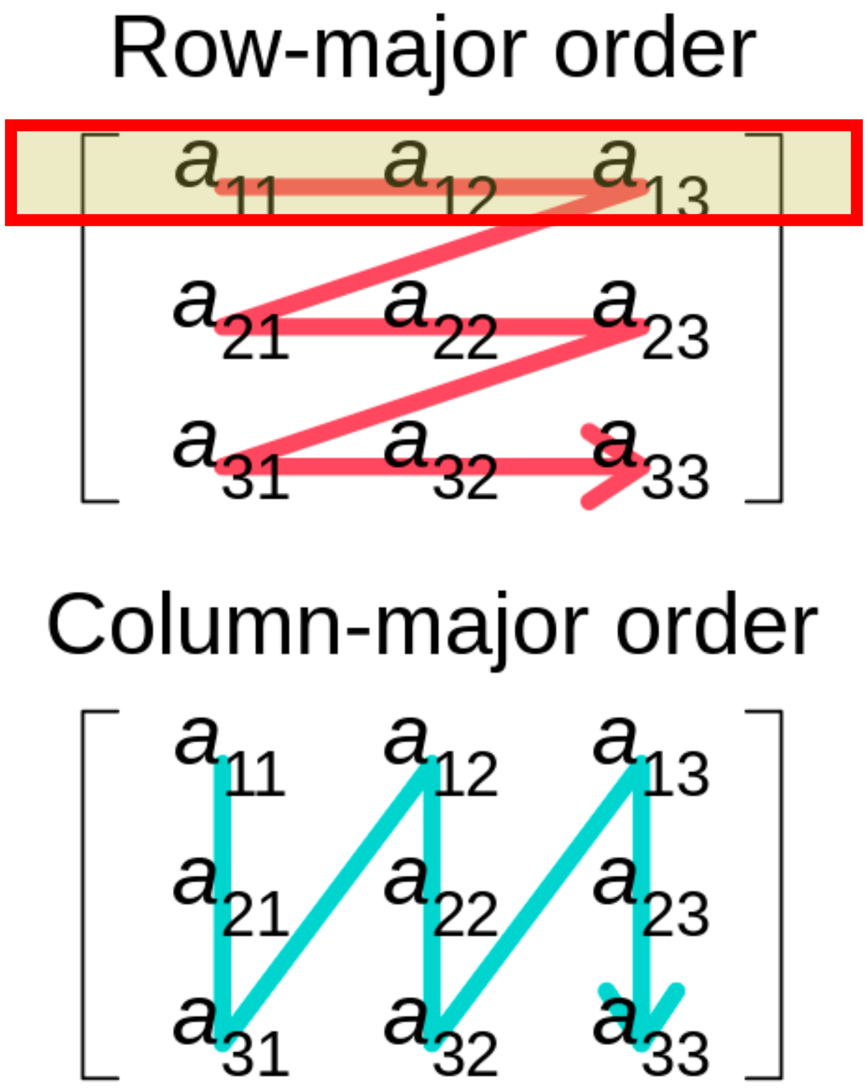
Loop Interchange

Original

```
for i from 0 to 10
  for j from 0 to 20
    a[i,j] = i + j
```

Transformed

```
for j from 0 to 20
  for i from 0 to 10
    a[i,j] = i + j
```



https://en.wikipedia.org/wiki/Loop_interchange

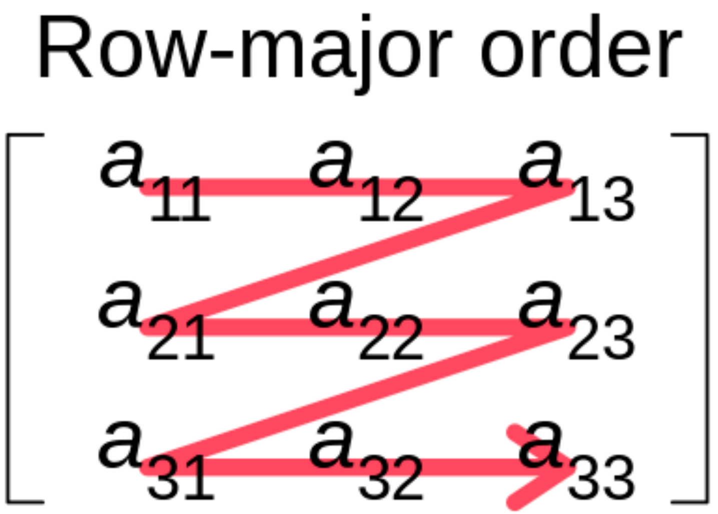
Loop Interchange

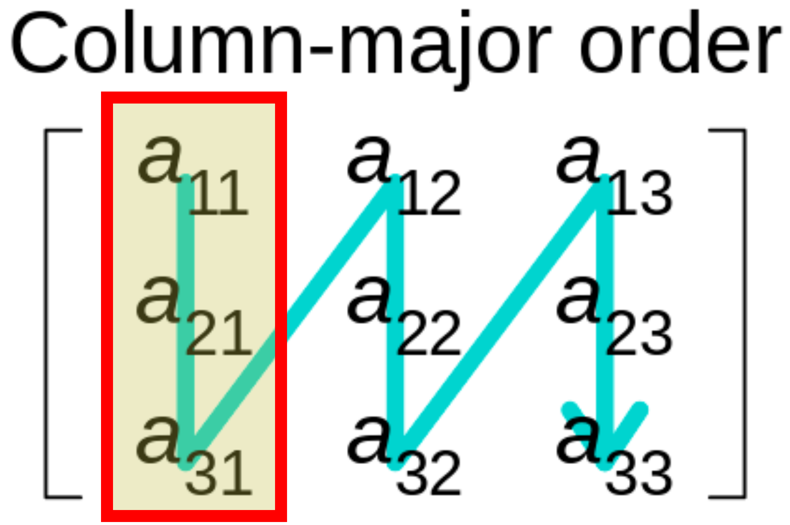
Original

```
for i from 0 to 10
  for j from 0 to 20
    a[i,j] = i + j
```

Transformed

```
for j from 0 to 20
  for i from 0 to 10
    a[i,j] = i + j
```





https://en.wikipedia.org/wiki/Loop_interchange

Additional Reading for Loop Transformations

- https://sites.cs.ucsb.edu/~tyang/class/240a13w/slides/LectureParallelization2s.pdf
- https://www.inf.ed.ac.uk/teaching/courses/copt/lecture-9.pdf
- https://www.cri.ensmp.fr/~tadonki/PaperForWeb/tadonki_loop.pdf

Loop Transformations

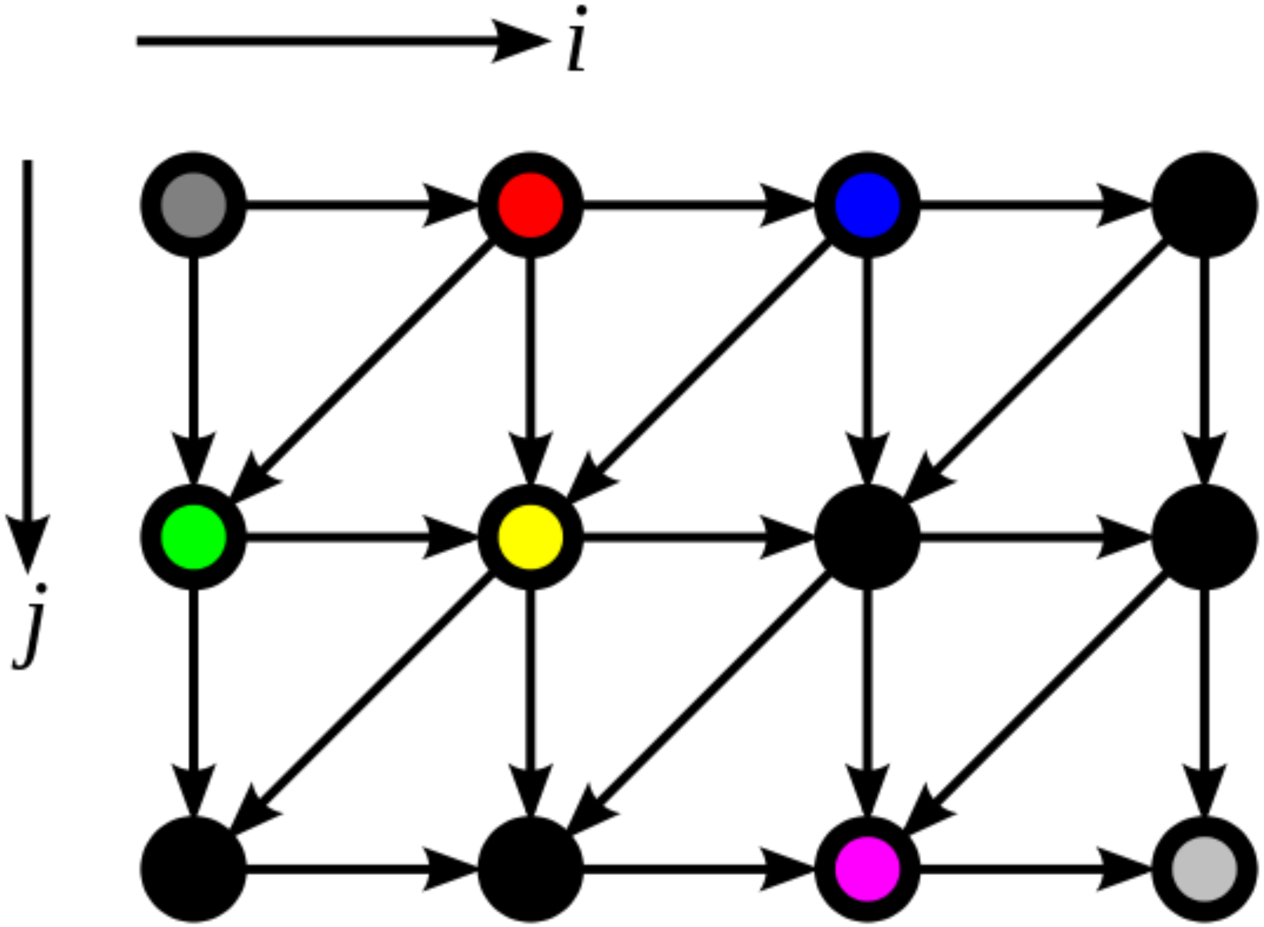
- Dependency Analysis
- Simple Transformations
- Loop Peeling
- Loop Fusion
- Loop Fission
- Loop Interchanging
- **Loop Skewing**
- Strip-Mining
- Loop Tiling

Loop Skewing

Original

```
#define ERR(x, y) (dst[x][y] - src[x][y])

void dither(unsigned char** src, unsigned char** dst, int w, int h)
{
    int i, j;
    for (j = 0; j < h; ++j) {
        for (i = 0; i < w; ++i) {
            int v = src[i][j];
            if (i > 0)
                v -= ERR(i - 1, j) / 2;
            if (j > 0) {
                v -= ERR(i, j - 1) / 4;
                if (i < w - 1)
                    v -= ERR(i + 1, j - 1) / 4;
            }
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```



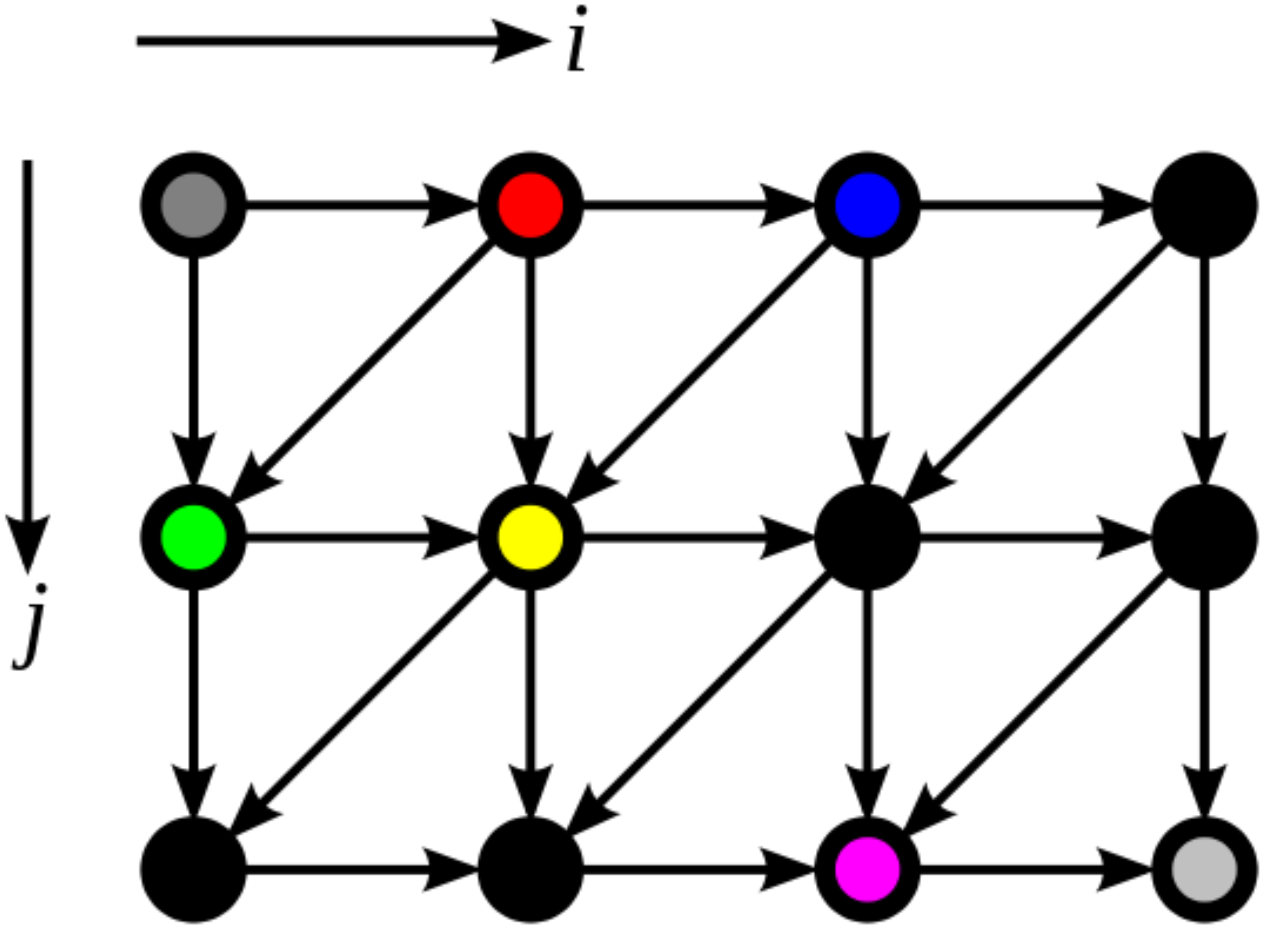
https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Original

```c
#define ERR(x, y) (dst[x][y] - src[x][y])

void dither(unsigned char** src, unsigned char** dst, int w, int h)
{
    int i, j;
    for (j = 0; j < h; ++j) {
        for (i = 0; i < w; ++i) {
            int v = src[i][j];
            if (i > 0)
                v -= ERR(i - 1, j) / 2;
            if (j > 0) {
                v -= ERR(i, j - 1) / 4;
                if (i < w - 1)
                    v -= ERR(i + 1, j - 1) / 4;
            }
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

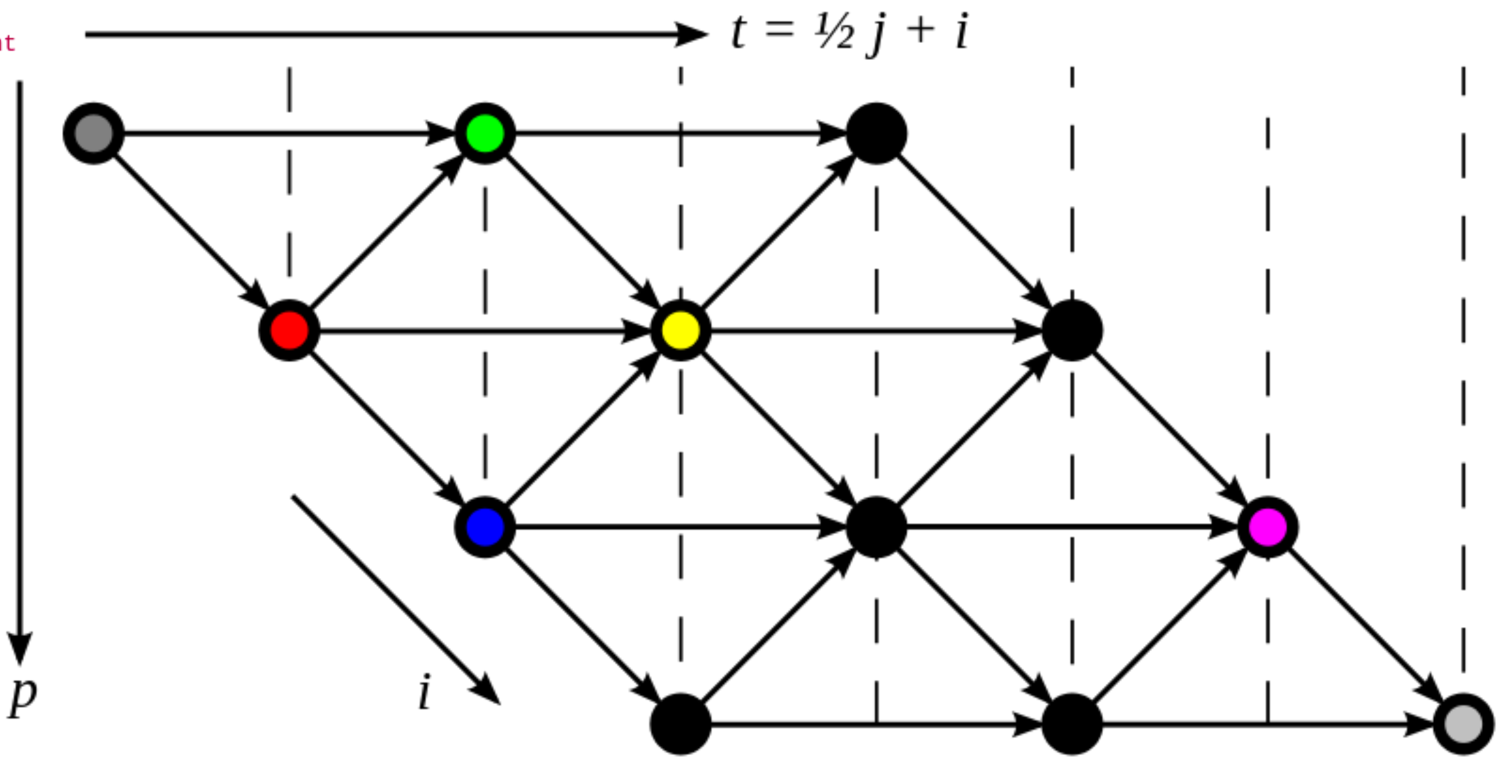


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

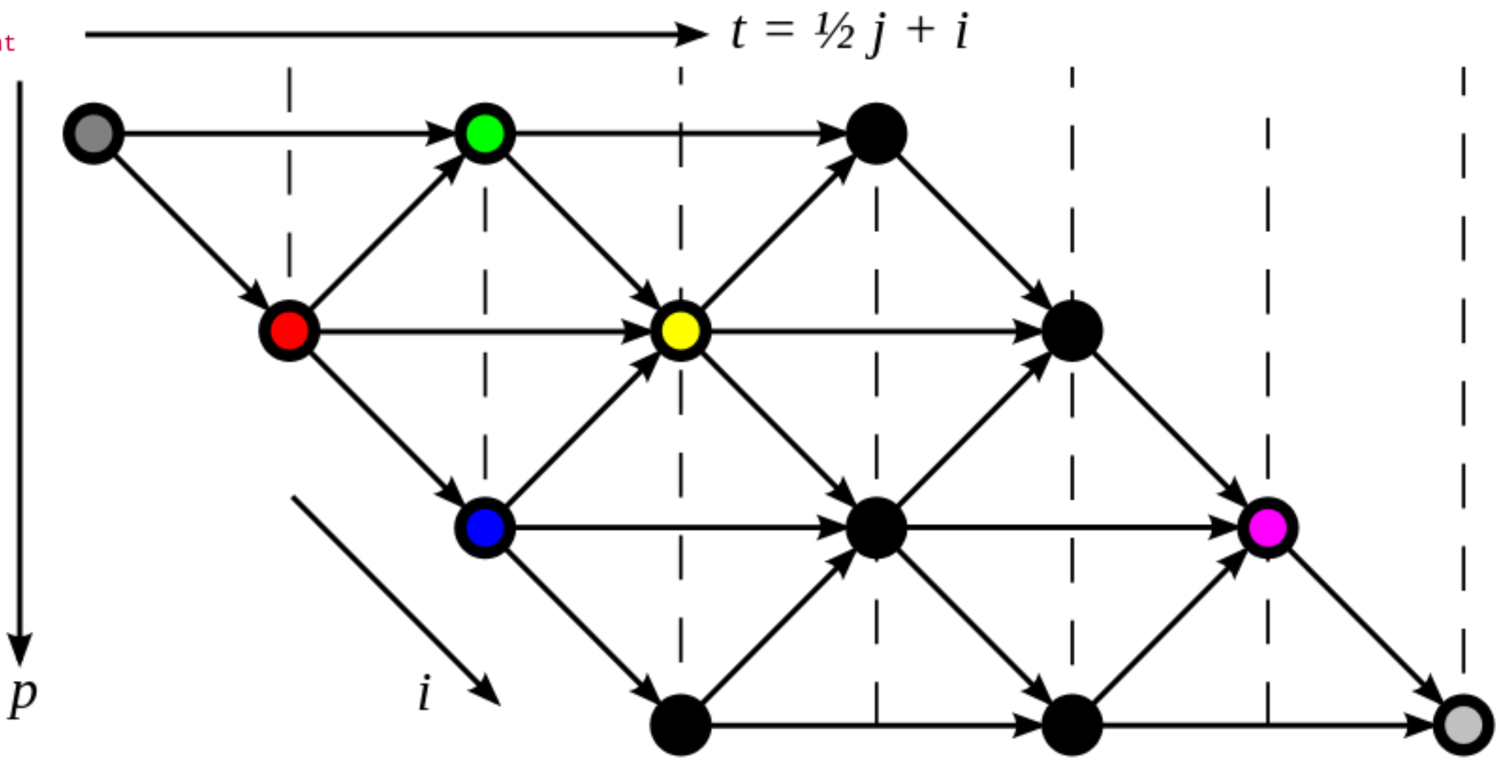


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

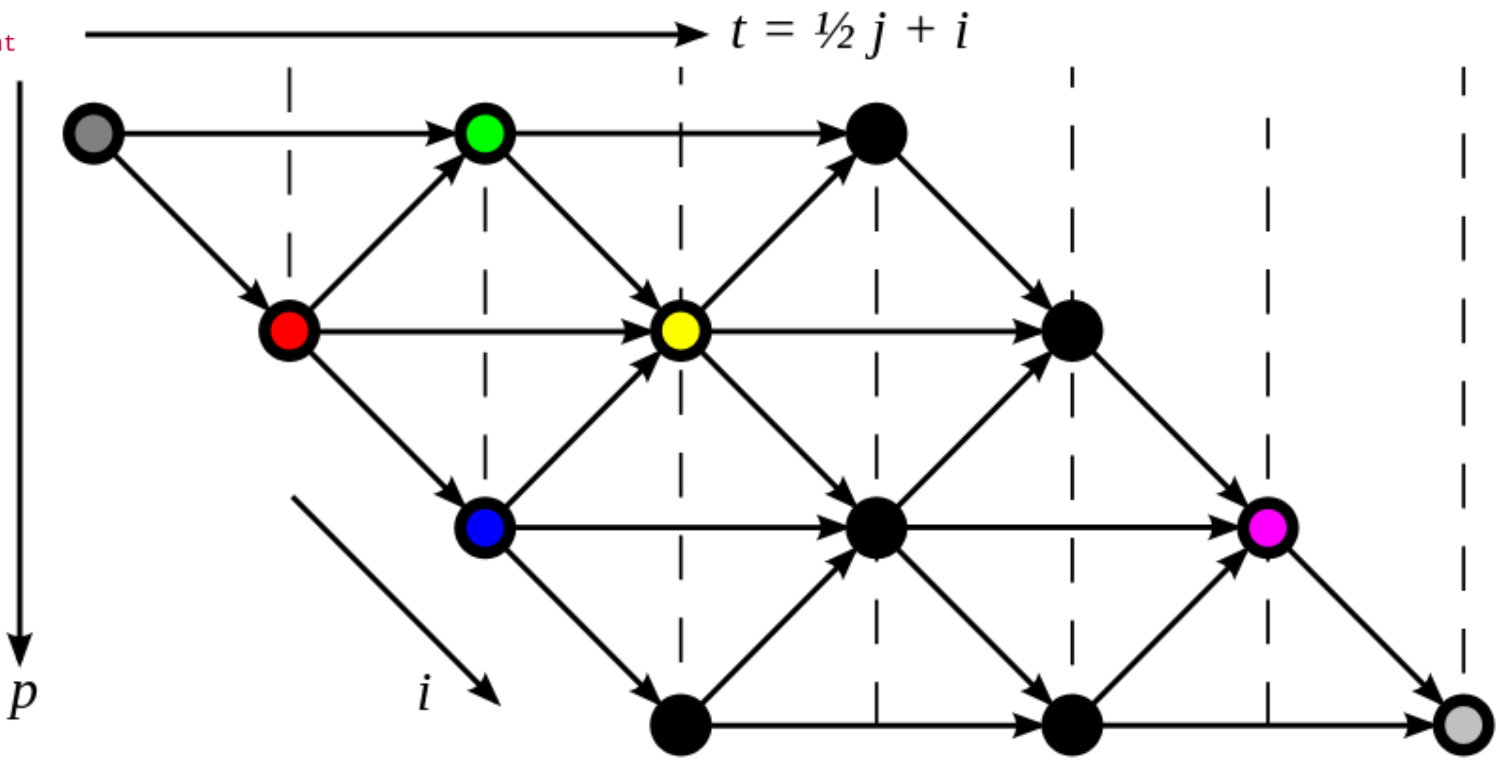


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

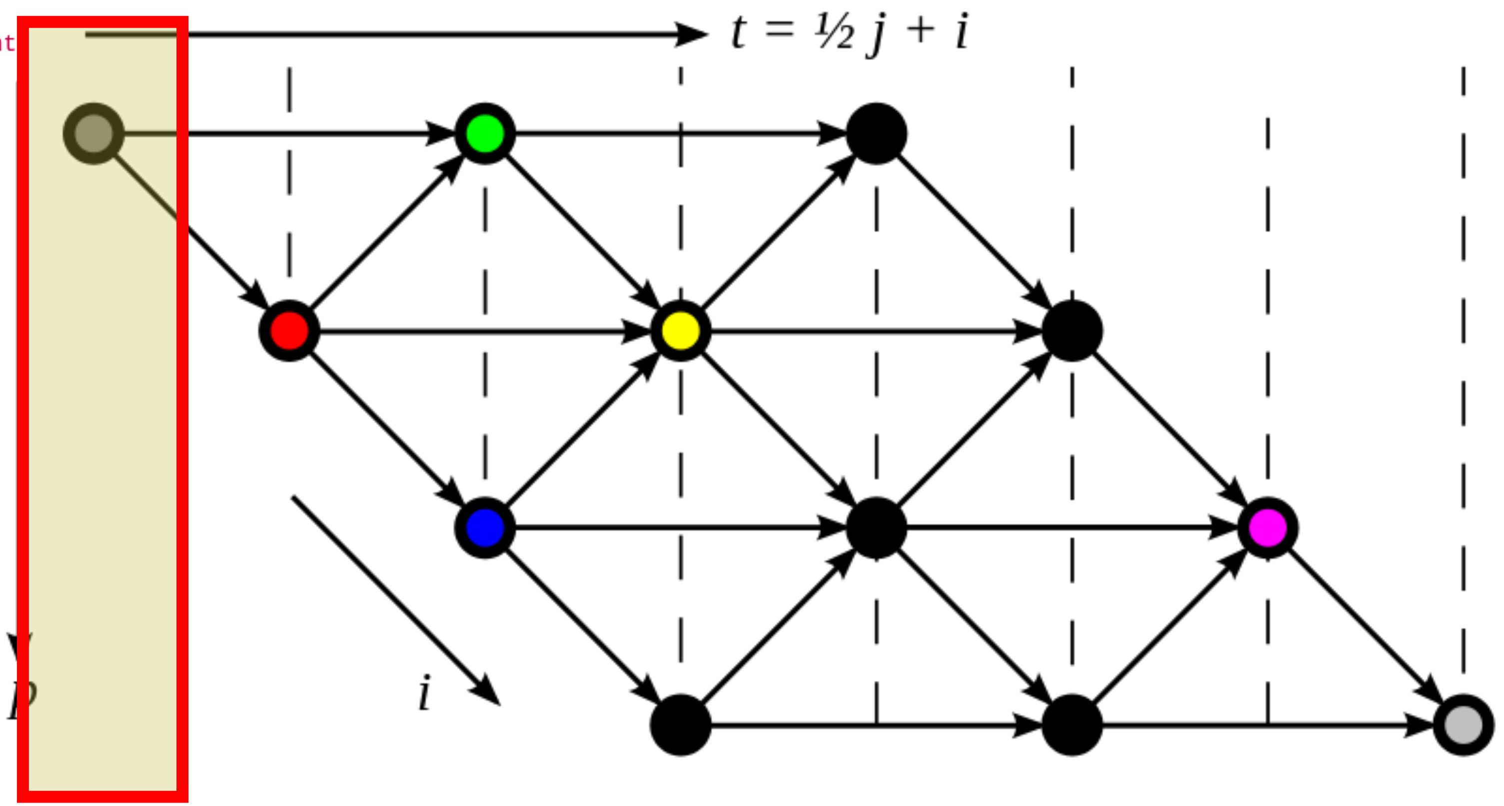


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

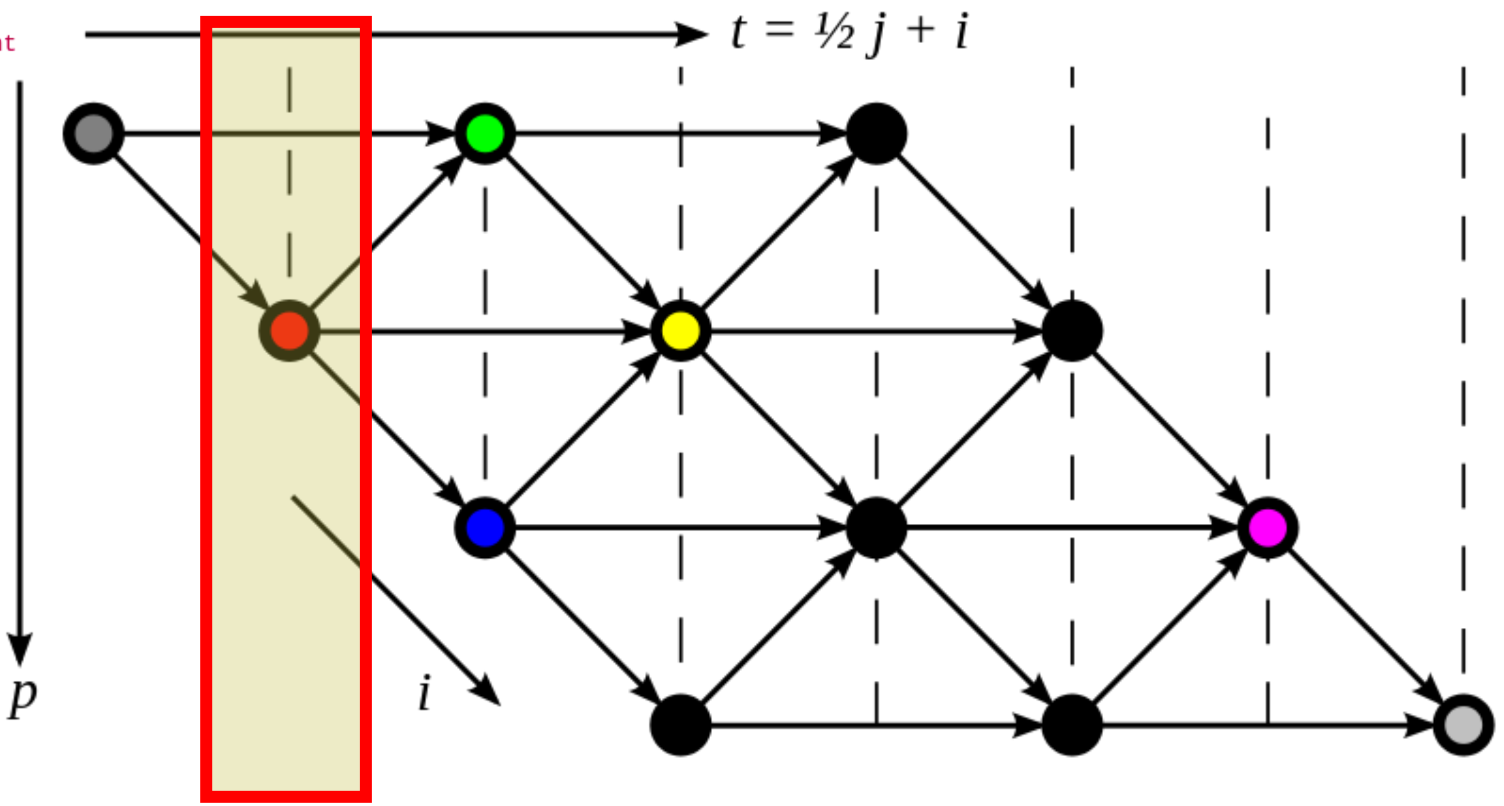


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

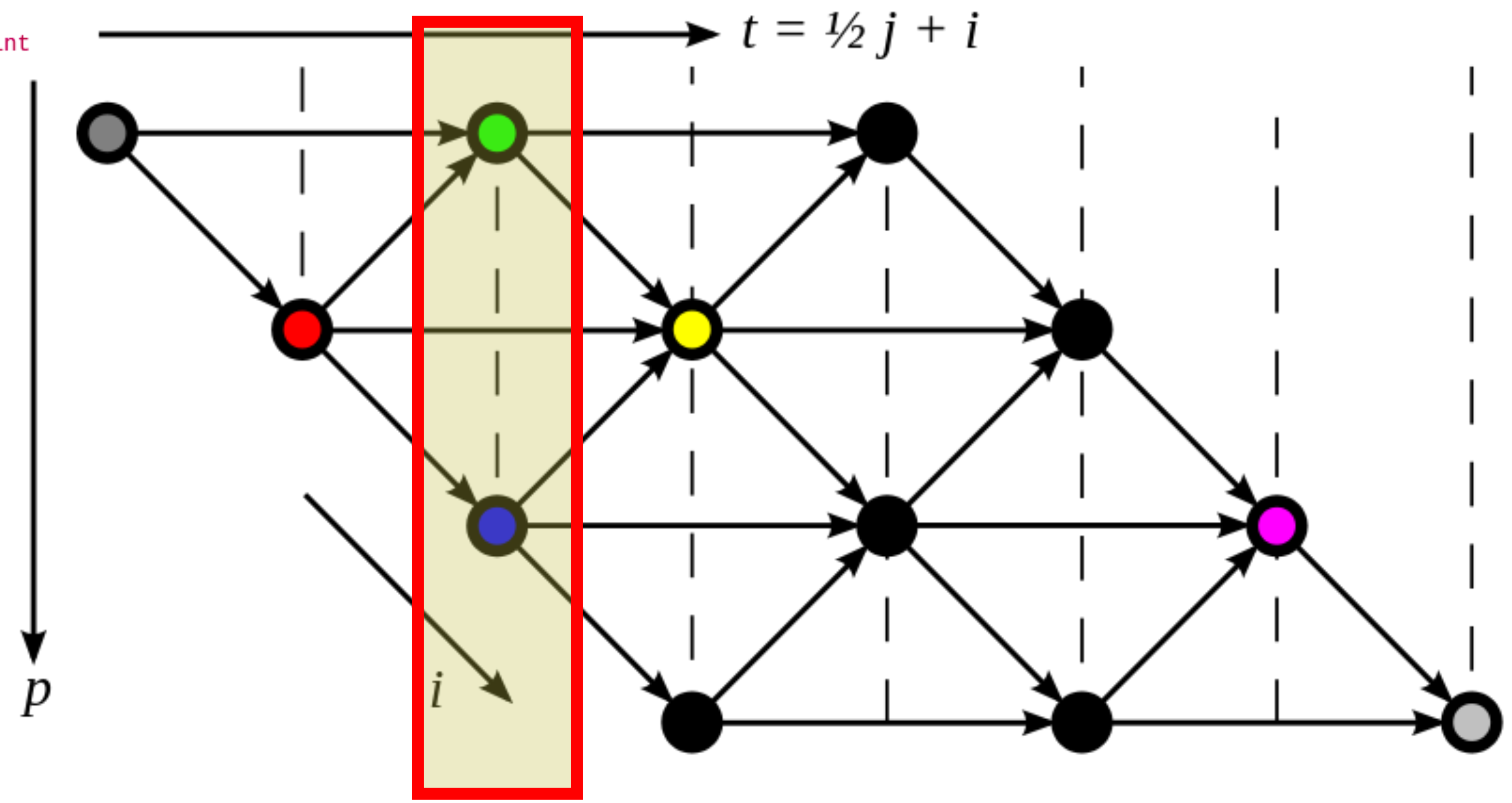


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

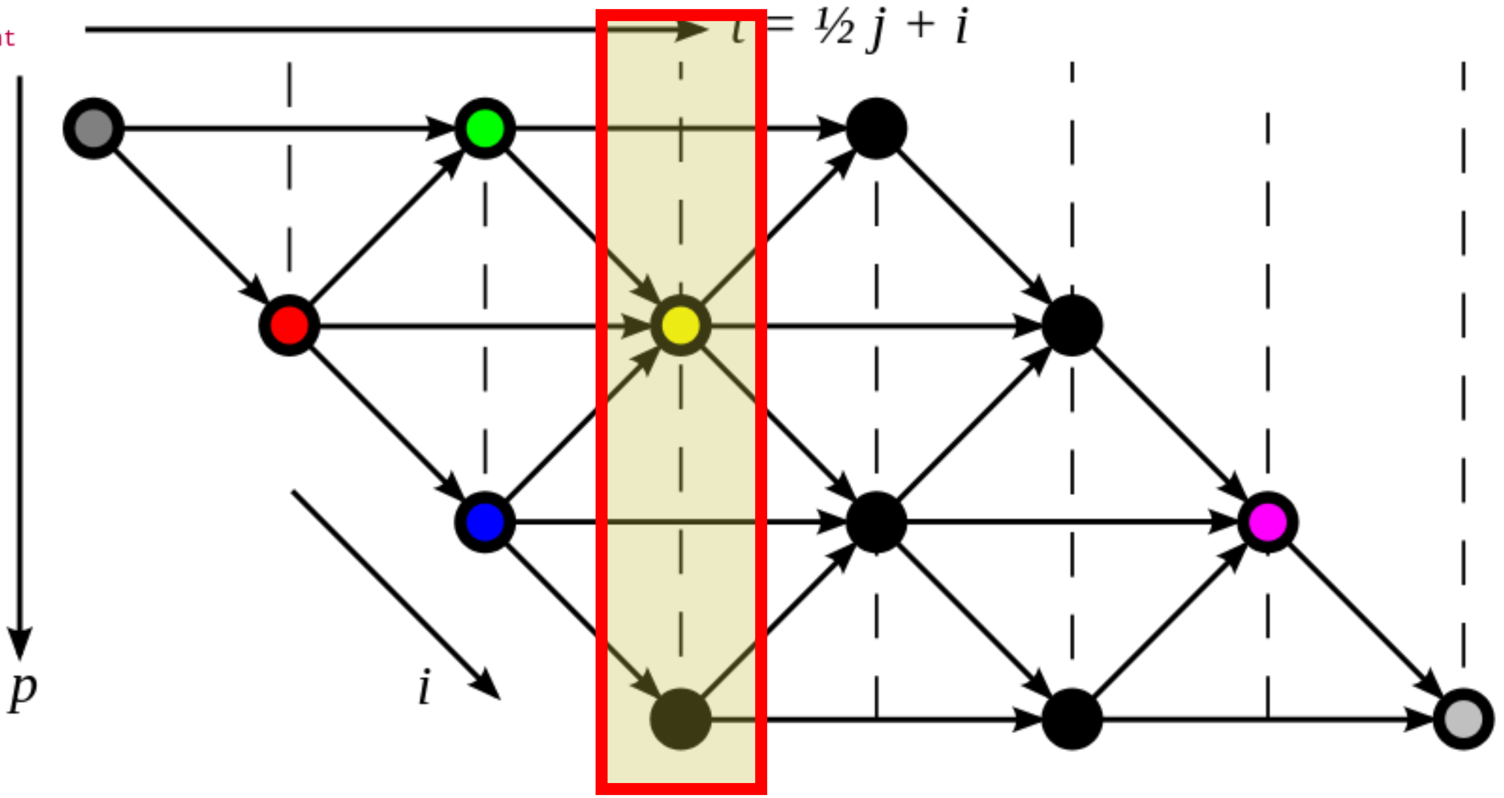


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

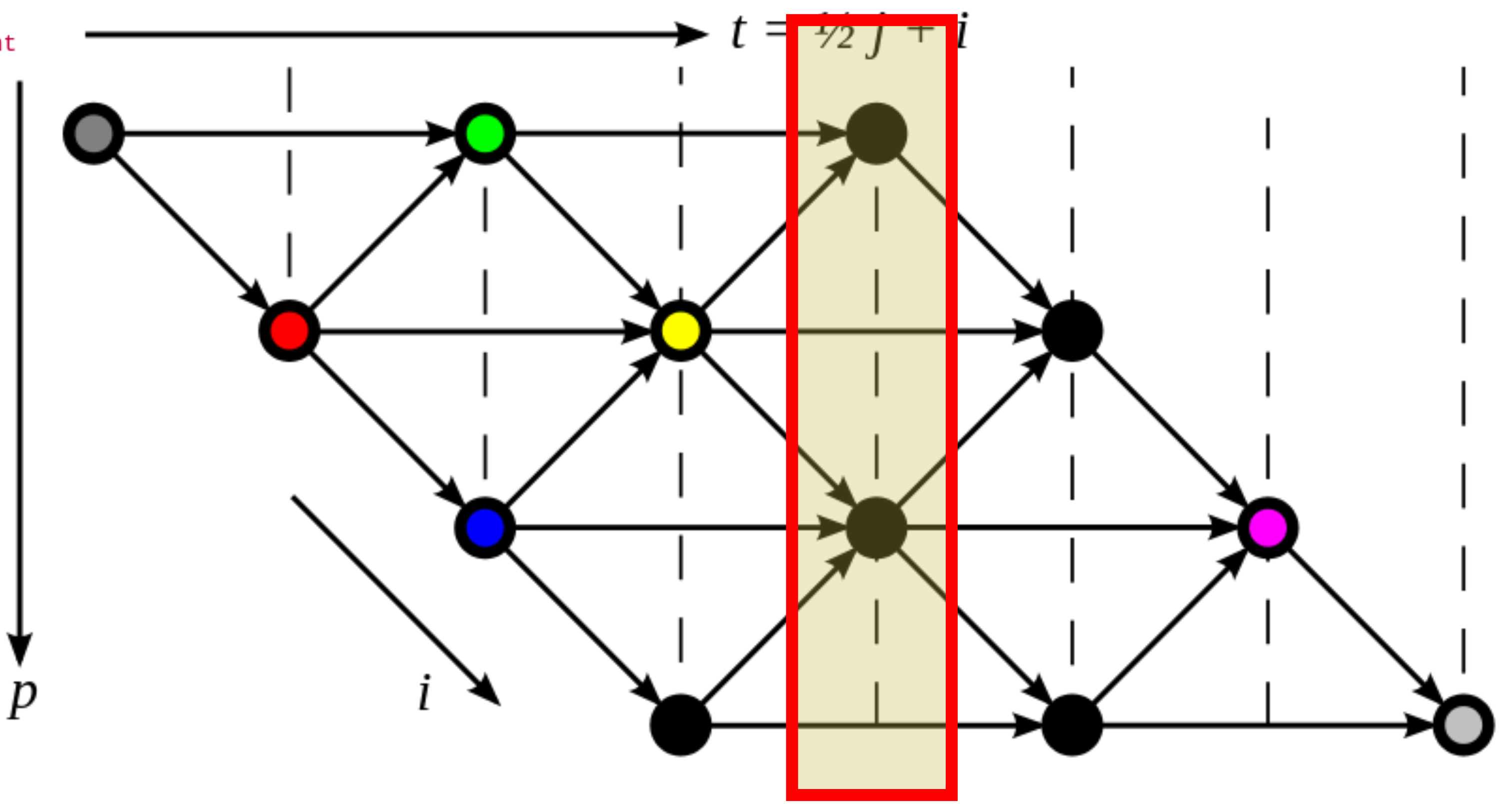


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

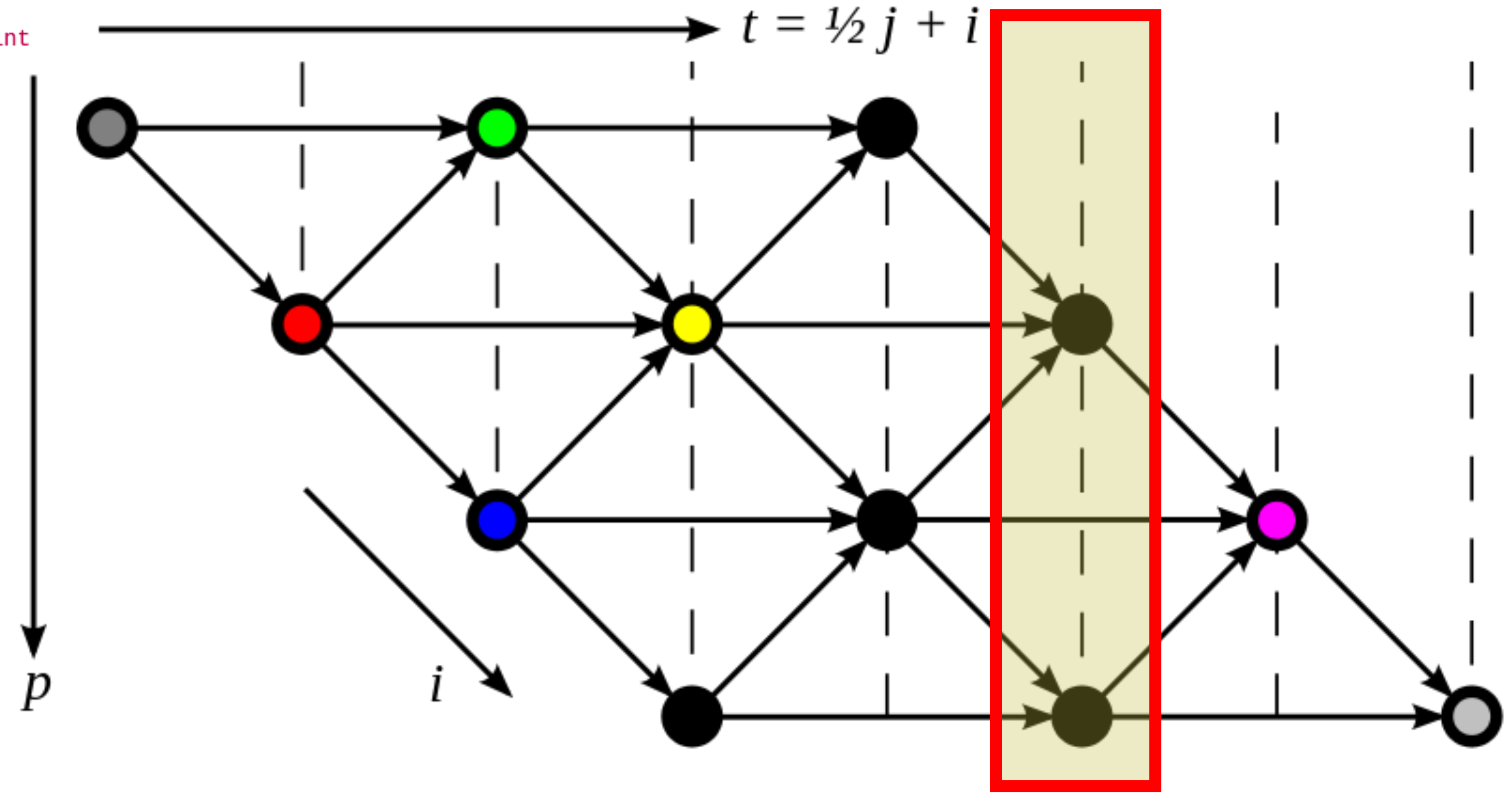


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
 {
     int t, p;
     for (t = 0; t < (w + (2 * h)); ++t) {
         int pmin = max(t % 2, t - (2 * h) + 2);
         int pmax = min(t, w - 1);
         for (p = pmin; p <= pmax; p += 2) {
             int i = p;
             int j = (t - p) / 2;
             int v = src[i][j];
             if (i > 0)
               v -= ERR(i - 1, j) / 2;
             if (j > 0)
               v -= ERR(i, j - 1) / 4;
             if (j > 0 && i < w - 1)
               v -= ERR(i + 1, j - 1) / 4;
             dst[i][j] = (v < 128) ? 0 : 255;
             src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
         }
     }
 }
```

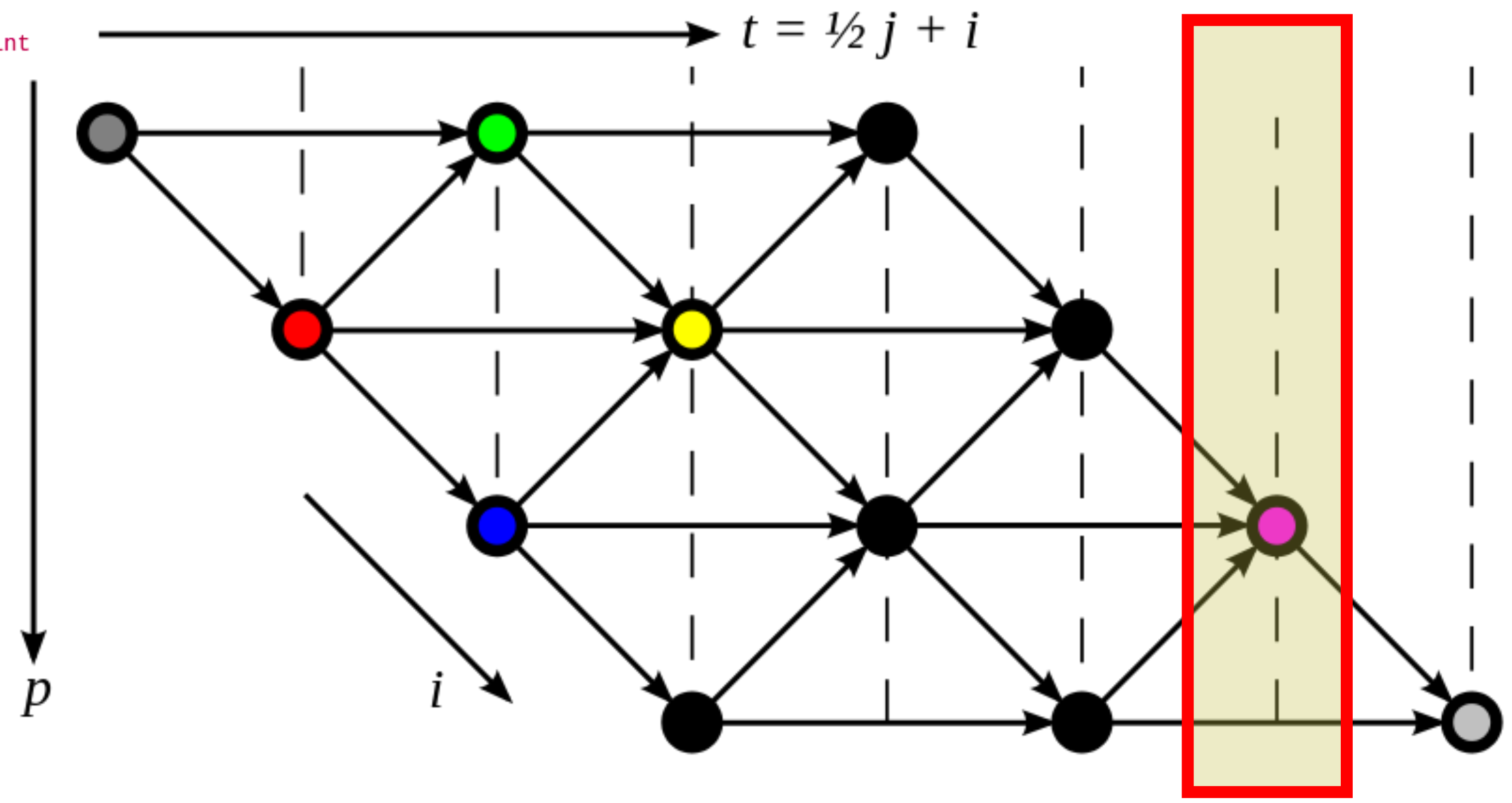


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

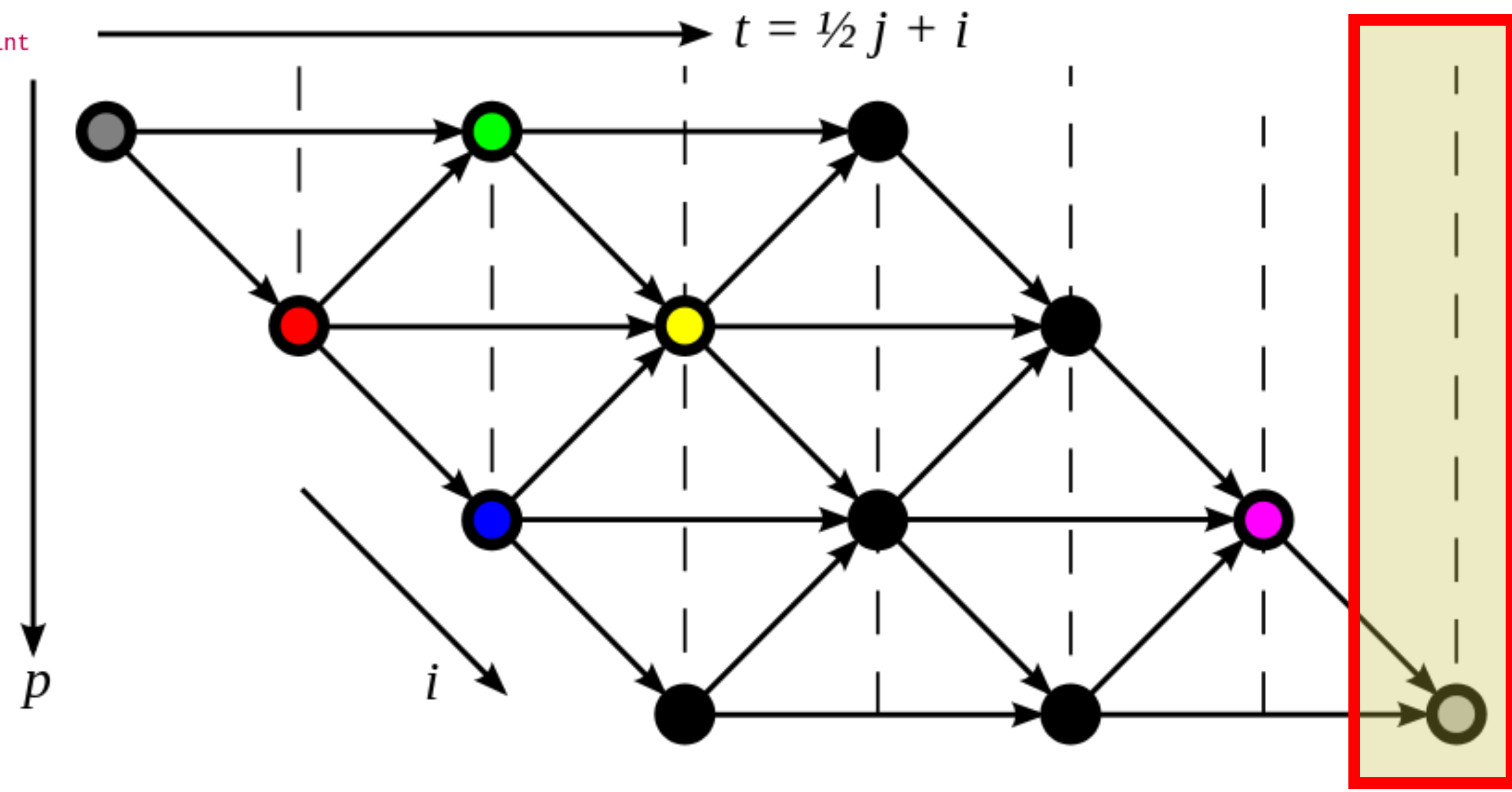


https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Transformed

```
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```



https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Original

```c
#define ERR(x, y) (dst[x][y] - src[x][y])

void dither(unsigned char** src, unsigned char** dst, int w, int h)
{
    int i, j;
    for (j = 0; j < h; ++j) {
        for (i = 0; i < w; ++i) {
            int v = src[i][j];
            if (i > 0)
                v -= ERR(i - 1, j) / 2;
            if (j > 0) {
                v -= ERR(i, j - 1) / 4;
                if (i < w - 1)
                    v -= ERR(i + 1, j - 1) / 4;
            }
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

Transformed

```c
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Original

```c
#define ERR(x, y) (dst[x][y] - src[x][y])

void dither(unsigned char** src, unsigned char** dst, int w, int h)
{
    int i, j;
    for (j = 0; j < h; ++j) {
        for (i = 0; i < w; ++i) {
            int v = src[i][j];
            if (i > 0)
                v -= ERR(i - 1, j) / 2;
            if (j > 0) {
                v -= ERR(i, j - 1) / 4;
                if (i < w - 1)
                    v -= ERR(i + 1, j - 1) / 4;
            }
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

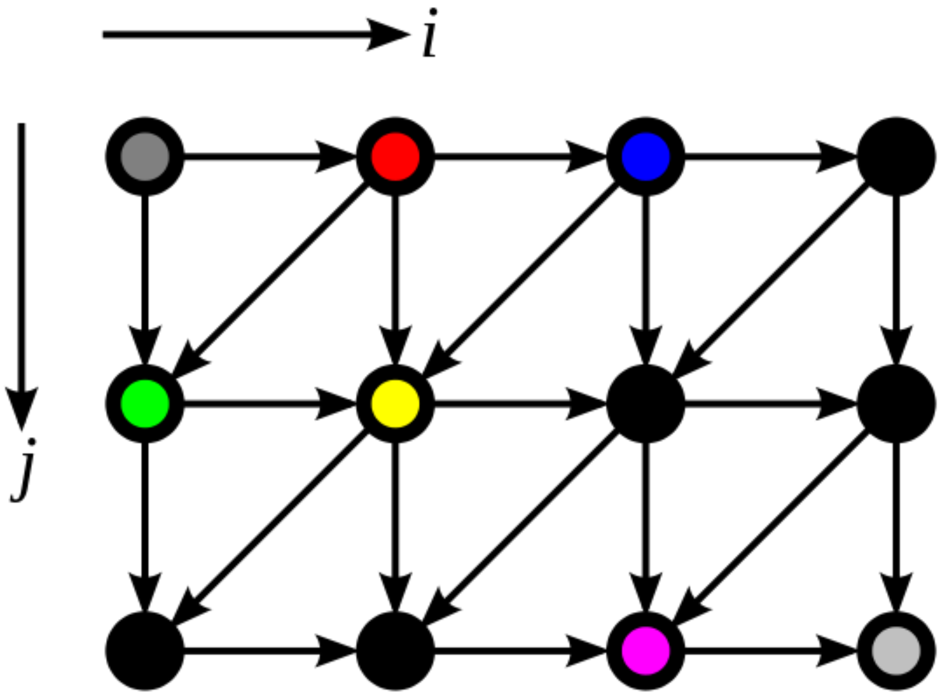
Transformed

```c
void dither_skewed(unsigned char **src, unsigned char **dst, int w, int
h)
{
    int t, p;
    for (t = 0; t < (w + (2 * h)); ++t) {
        int pmin = max(t % 2, t - (2 * h) + 2);
        int pmax = min(t, w - 1);
        for (p = pmin; p <= pmax; p += 2) {
            int i = p;
            int j = (t - p) / 2;
            int v = src[i][j];
            if (i > 0)
              v -= ERR(i - 1, j) / 2;
            if (j > 0)
              v -= ERR(i, j - 1) / 4;
            if (j > 0 && i < w - 1)
              v -= ERR(i + 1, j - 1) / 4;
            dst[i][j] = (v < 128) ? 0 : 255;
            src[i][j] = (v < 0) ? 0 : (v < 255) ? v : 255;
        }
    }
}
```

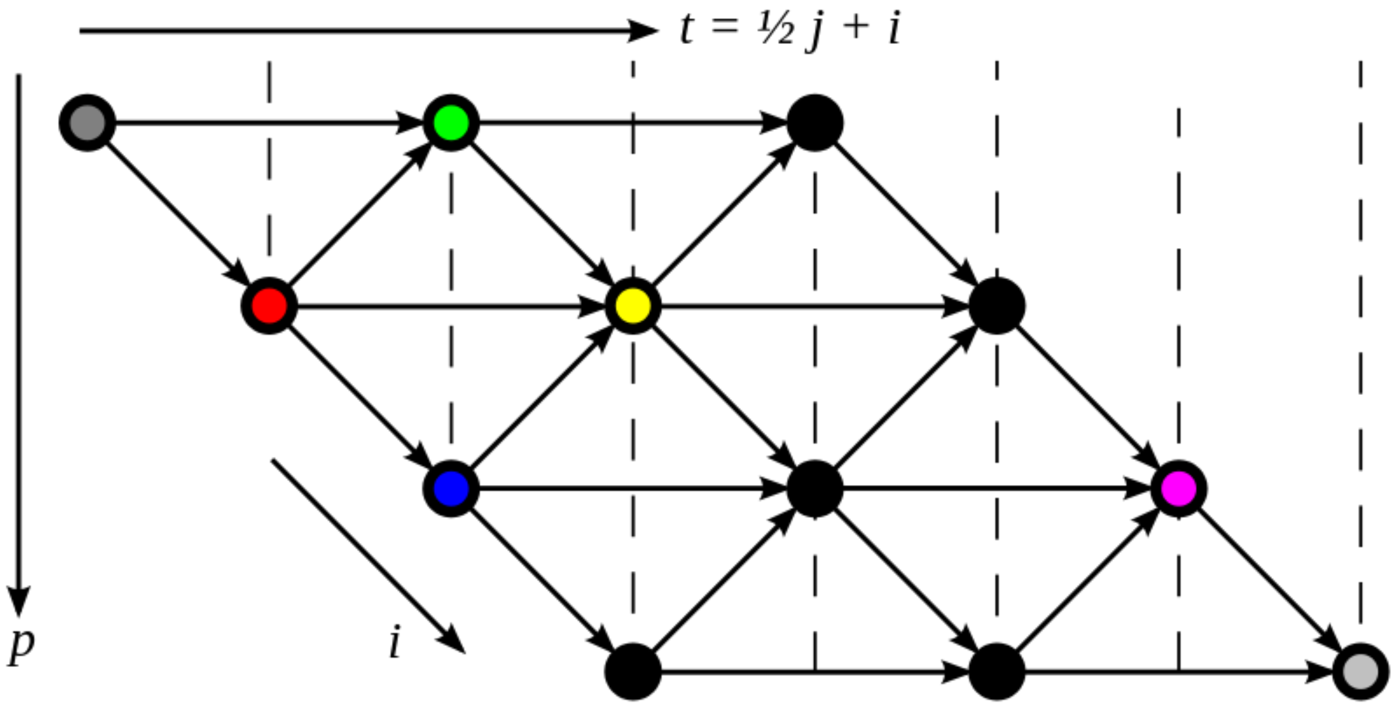
https://en.m.wikipedia.org/wiki/Polytope_model

Loop Skewing

Original



Transformed



https://en.m.wikipedia.org/wiki/Polytope_model

Loop Transformations

- Dependency Analysis
- Simple Transformations
- Loop Peeling
- Loop Fusion
- Loop Fission
- Loop Interchanging
- Loop Skewing
- Strip-Mining
- Loop Tiling

Strip Mining

Original

```
for i = 1 to N
   S
```

Transformed

```
for (i = 0; i < N; i += U)
   for (j = 0; j < U; j++)
      S(i + j)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Strip Mining

Original

```
for i = 1 to N
  S
```

Transformed

```
for (i = 0; i < N; i += U)
  for (j = 0; j < U; j++)
    S(i + j)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Strip Mining

Original

```
for i = 1 to N
  S
```

Transformed

```
for (i = 0; i < N; i += U)
  for (j = 0; j < U; j++)
    S(i + j)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Strip Mining

Original

```
for i = 1 to N
   S
```

Transformed

```
for (i = 0; i < N; i += U)
   for (j = 0; j < U; j++)
      S(i + j)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Strip Mining

Original

```
for i = 1 to N
   S
```

Transformed

```
for (i = 0; i < N; i += U)
   for (j = 0; j < U; j++)
      S(i + j)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Loop Transformations

- Dependency Analysis
- Simple Transformations
- Loop Peeling
- Loop Fusion
- Loop Fission
- Loop Interchanging
- Loop Skewing
- Strip-Mining
- Loop Tiling

Tiling (Strip Mining + Interchange

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Interchanged

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Tiling (Strip Mining + Interchange

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Interchanged

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Tiling (Strip Mining + Interchange

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Interchanged

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Tiling (Strip Mining + Interchange

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Interchanged

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Tiling (Strip Mining + Interchange

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Interchanged

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Tiling (Strip Mining + Interchange

Original

```
for i = 1 to N
  for j = 1 to M
    S(i, j)
```

Strip-mined

```
for i = 1 to N step S
  for ii = 1 to S
    for j = 1 to M step T
      for jj = 1 to T
        S(i + ii, j + jj)
```

Interchanged

```
for i = 1 to N step S
  for j = 1 to M step T
    for ii = 1 to S
      for jj = 1 to T
        S(i + ii, j + jj)
```

https://compilers.cs.uni-saarland.de/teaching/cc/2017/slides/loops.pdf

Loop Transformations

- Dependency Analysis
- Simple Transformations
- Loop Peeling
- Loop Fusion
- Loop Fission
- Loop Interchanging
- Loop Skewing
- Strip-Mining
- Loop Tiling

Amdahl's Law

$$S = \frac{r+k}{r+k/p} = \frac{1}{r+(1-r)/p}$$

$$\lim_{p \to \infty} S = \frac{1}{r}$$

- r is the fraction of a serial program that remains unparallelized
- k is the fraction of a serial program that has been parallelized
- $r + k = 1$ for a serial run-time
- p is the number of processors

Amdahl's Law focuses on a fixed amount of workload, which has a constant r.

However, in practice, a supercomputer is used not to speed up a fixed amount of workload, but to handle a larger amount of workload within a fixed amount of time.

Gustafson's law

$$S = \frac{h + w \times p}{h + w} = h + (1 - h)p$$

$$\lim_{p \to \infty} S = \infty$$

- h is the fraction of a parallel program that remained serial. Assume that h does not increase with p.
- w is the fraction of a parallel program that is parallelized. Assume that this fraction has linear scalability with p.
- $h + w = 1$ for a parallel run-time
- p is the number of processors

- Amdahl's Law focuses on speeding up a fixed amount of serial workload using more and more processors
- Gustafson's law focuses on solving larger and larger problems within a fixed amount of parallel run-time using more and more processors

Amdahl's Law

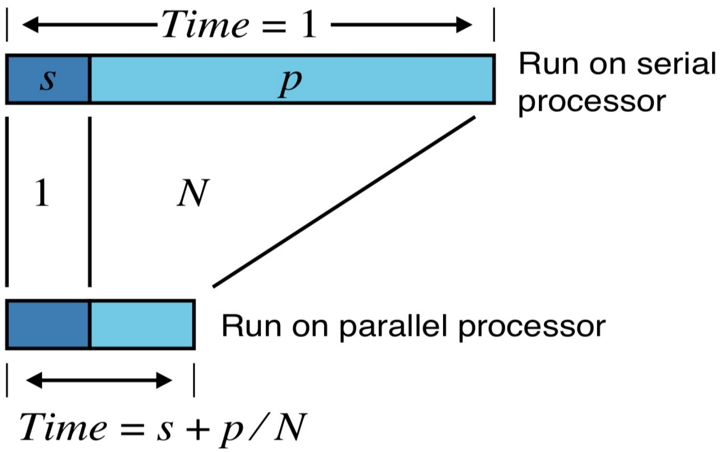


Figure 2a. Fixed-Size Model: $Speedup = 1 / (s + p / N)$

Gustafson's law

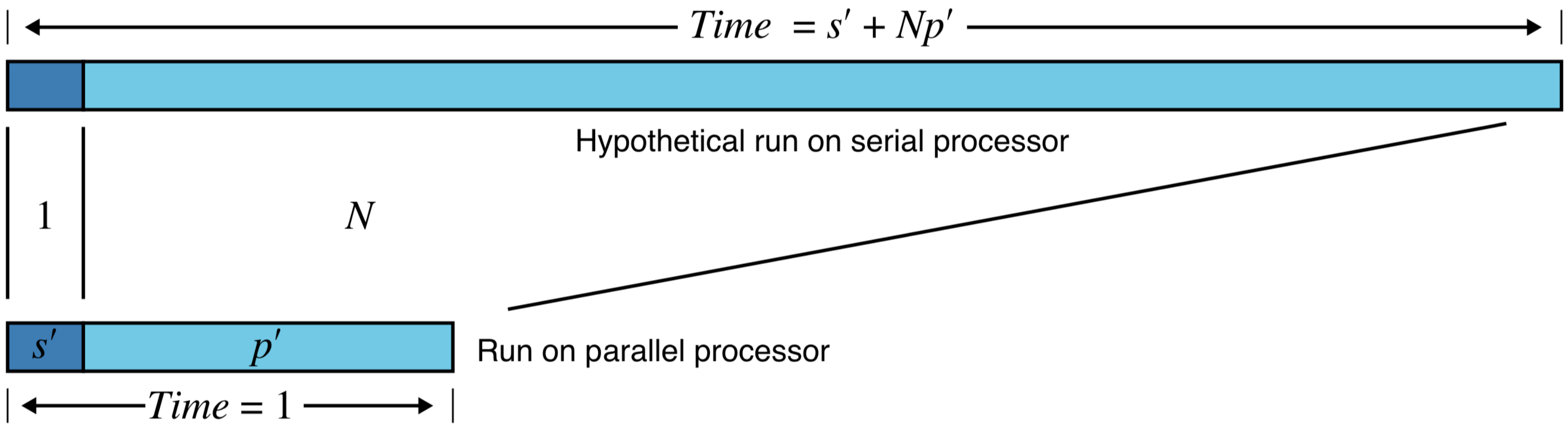


FIGURE 2b. Scaled-Size Model: $Speedup = s + Np$

Reevaluating Amdahl's Law. John L. Gustafson

http://www.johngustafson.net/pubs/pub13/amdahl.pdf

Scalability

- Programs that can maintain a constant efficiency without increasing the problem size are said to be strongly scalable.
- Programs that can maintain a constant efficiency if the problem size increases at the same rate as the number of processes are said to be weakly scalable.

size of the problem

Speedup

number of processes

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.0	1.0	1.0	1.0	1.0
2	1.8	1.9	1.9	1.9	2.0
4	2.1	3.1	3.6	3.9	3.9
8	2.4	4.8	6.5	7.5	7.9
16	2.4	6.2	10.8	14.2	15.5

Scalability

- Programs that can maintain a constant efficiency without increasing the problem size are said to be strongly scalable.
- Programs that can maintain a constant efficiency if the problem size increases at the same rate as the number of processes are said to be weakly scalable.

size of the problem

Efficiency

number of processes

comm_sz	Order of Matrix				
	1024	2048	4096	8192	16,384
1	1.00	1.00	1.00	1.00	1.00
2	0.89	0.94	0.97	0.96	0.98
4	0.51	0.78	0.89	0.96	0.98
8	0.30	0.61	0.82	0.94	0.98
16	0.15	0.39	0.68	0.89	0.97