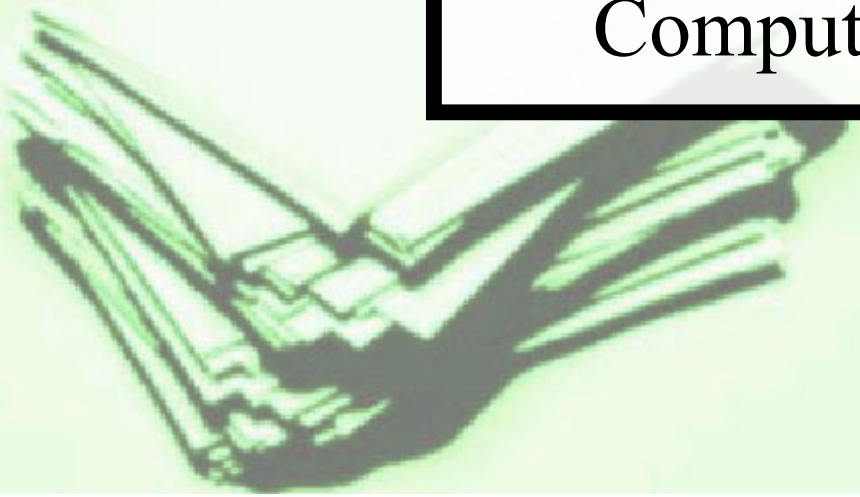
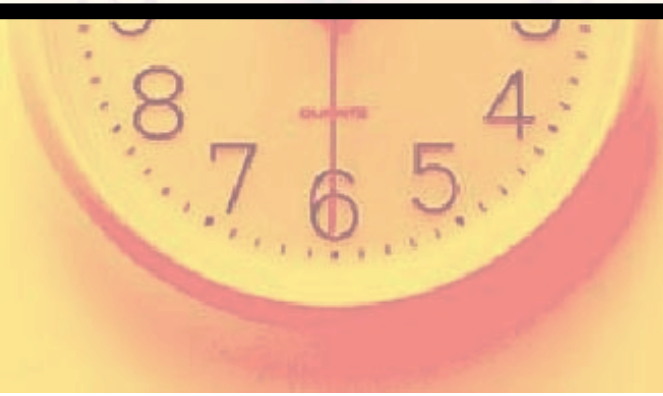
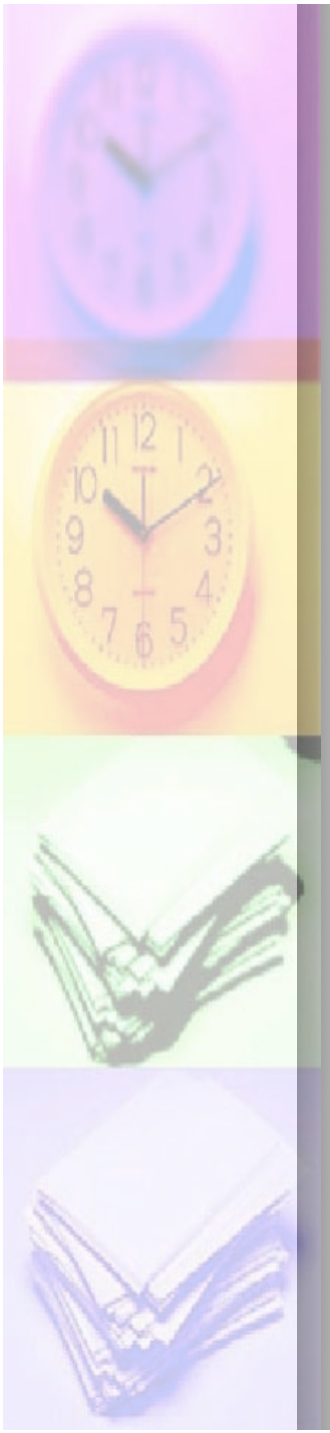
CS560: Foundations of the Polyhedral Model

Sanjay Rajopadhye

Computer Science Dept. CSU

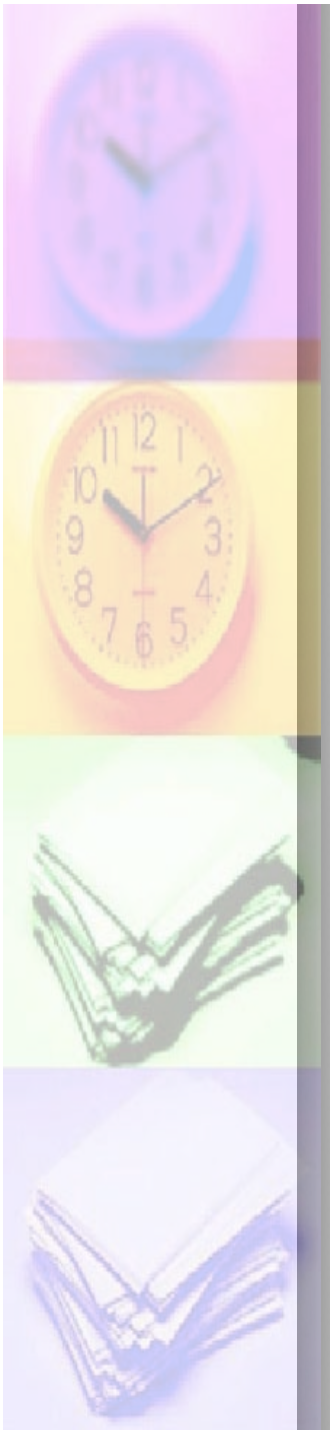






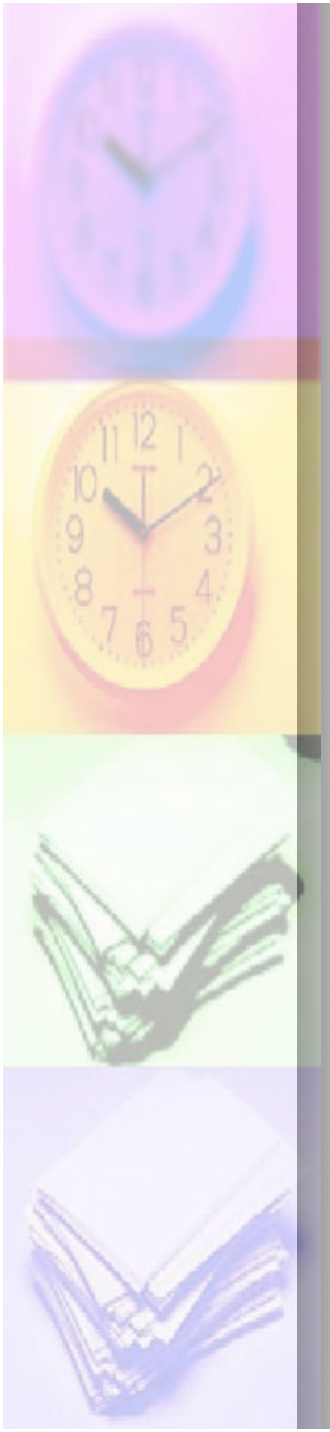
Parallelism is here and everywhere

- Parallel Programming is hard
- "Manycores imply the end of *La-Z-Boy Programming*." David Patterson, 2006/2007
- Learning to write efficient parallel programs will get you a good job
  - but it won't change the world
- We will reap the benefits of the new Moore's Law, only with the return of *La-Z-Boy Programming*."



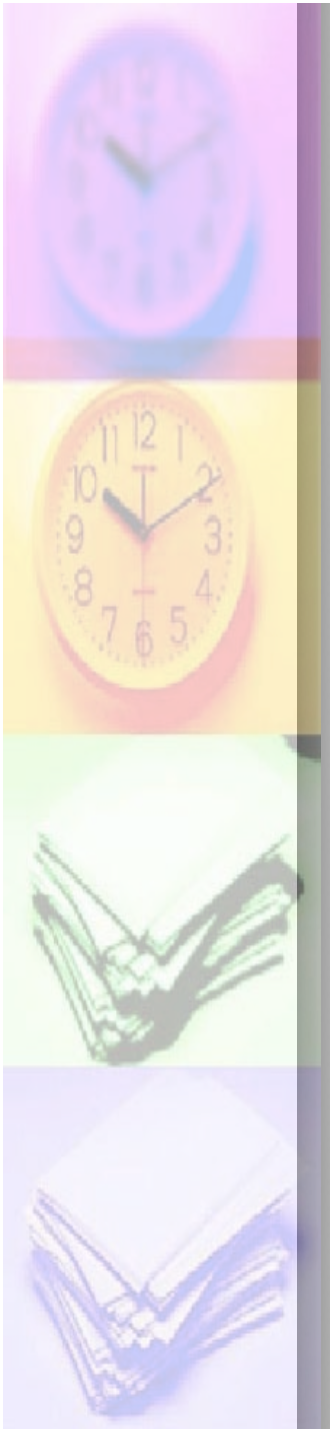
Class Objectives

- Short term: Learn to write highly tuned "code" for emerging target architectures – GPUs, many-cores, FPGAs, etc.
- Medium term: Do this so that you are not "blindsided" by the next hot new architectural paradigm – learn principles, not skills.
- Long Term: Better still, do this automatically, so that even such a skilled practitioner is rendered obsolete – be inspired by the research questions.



Class Approach & Outline (1/3)

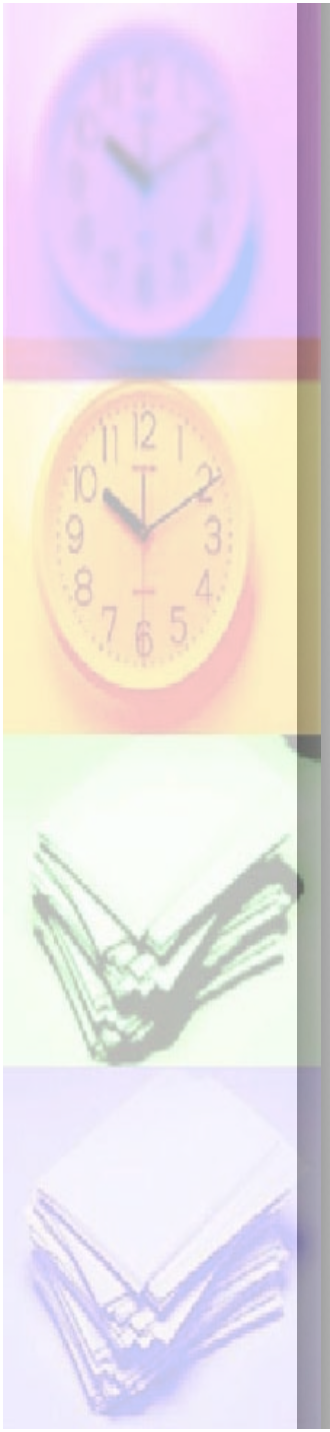
- Big picture first.
  - Polyhedral Equations as programs
  - Where do they come from?
  - Relation to “conventional” loop programs
- Implementing/compiling equations
  - Schedule
  - Processor Allocation
  - Memory allocation
- But where’s the parallelism?
- Too much parallelism?



Class Approach & Outline (2/3)

Five main Assignments + project:

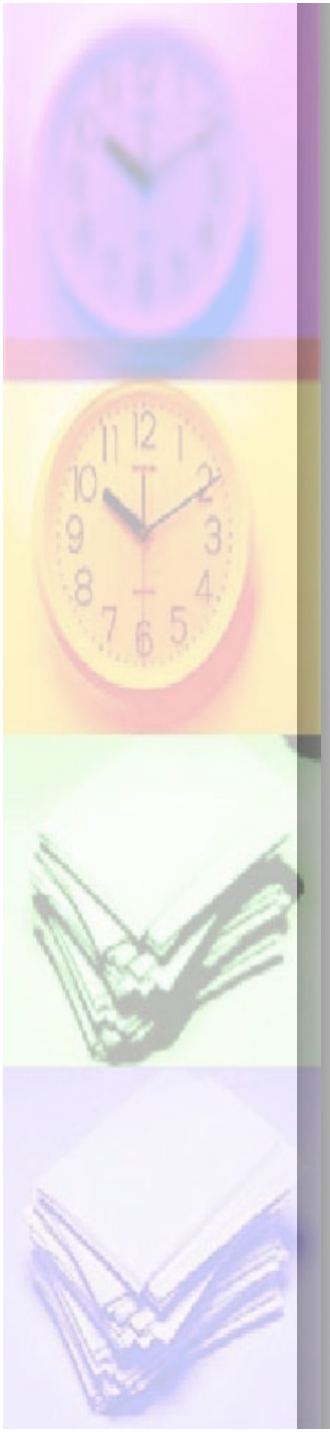
- Writing and executing equations
- Deriving equations from loop programs
- “Manipulating” and transforming polyhedra
- Scheduled (sequential) code generation
- Parallel code generation
- Tiling
  - manual
  - systematic
  - automatic



Class Approach & Outline (3/3)

Break from the past

- Use of tools – prototype research – tools (developed here and elsewhere)
  - AlphaZ
  - FADA
- Driven by the assignments
  - You learn by doing



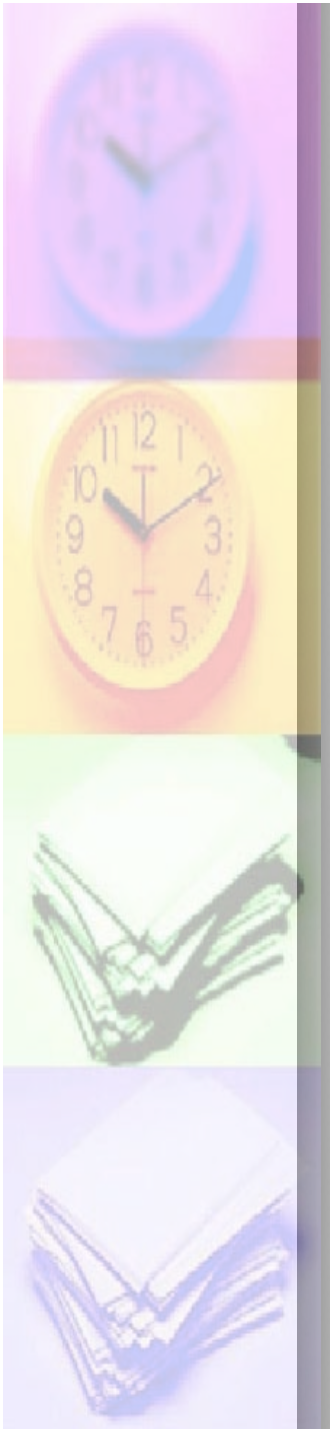
Three simple examples

- Write a program to calculate:

$$Y[i] = \sum_{j=1}^{i} X[j]$$

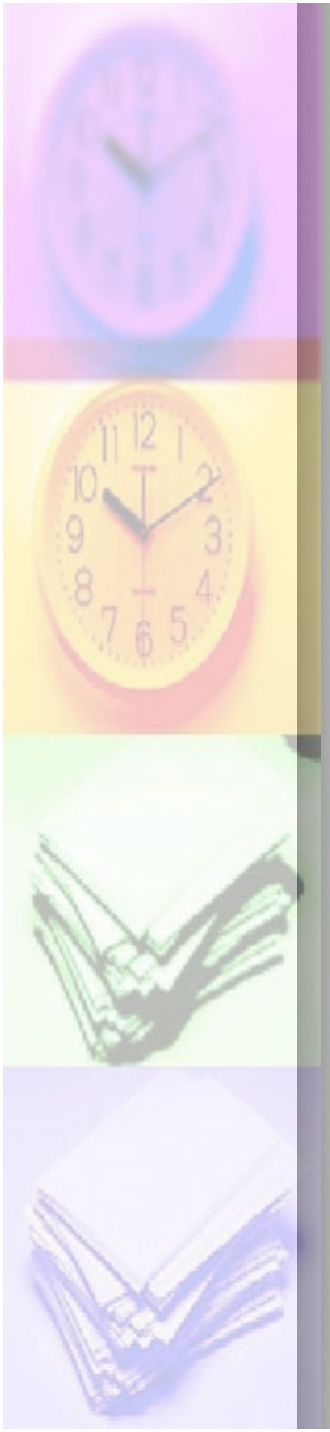
$$Y[i] = \max_{j=1}^{i} X[j]$$

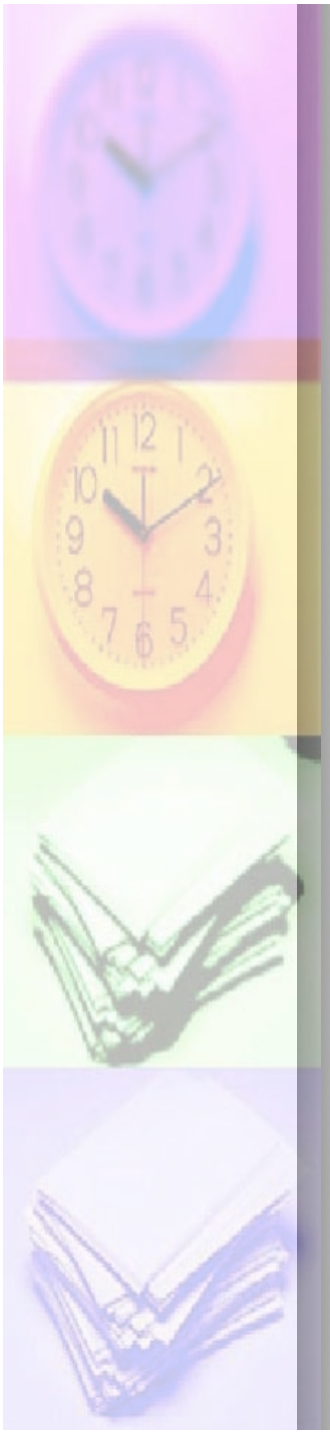
$$Y[i] = \max_{j=i}^{i+M} X[j]$$



More examples (from notes)

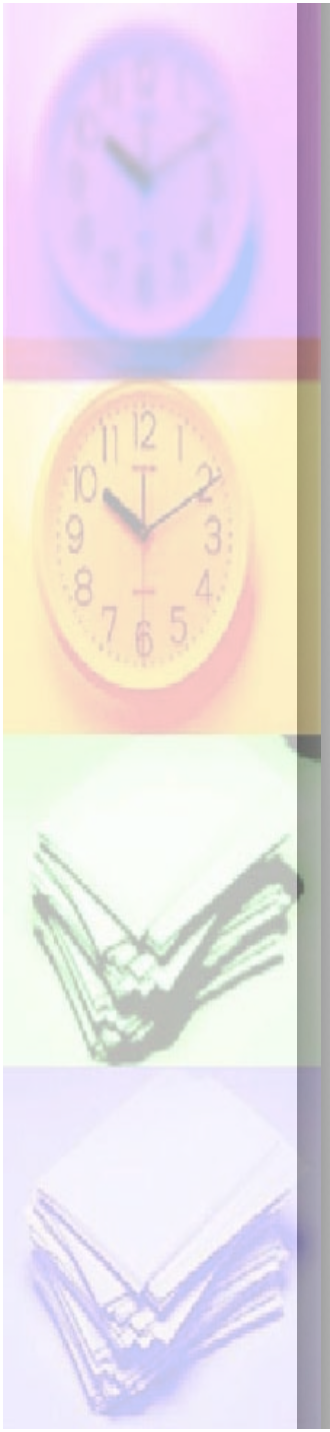
- Forward Substitution
- Cholesky Decomposition
- Dense linear algebra
- Dynamic Programming





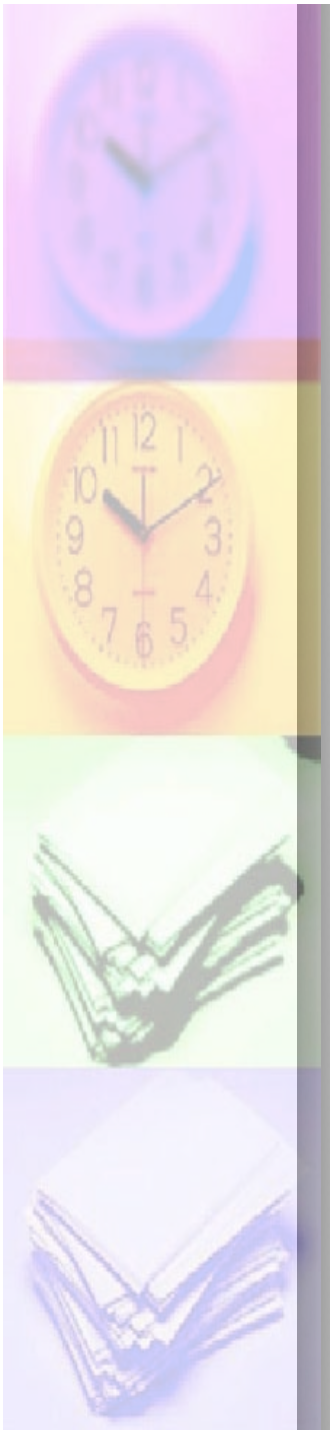
Pros and cons of Equations

- 90% of time is spent in 10% of the code
  - Polyhedral loops are performance-critical
  - So what?
- 90% of programmer time is spent in 90% of the code.



Who REALLY needs such specs?

- End programmers
  - but they may just as well write simple loop programs.
- Performance geeks (application tuners)
  - but need many additional tools
- Compilers
  - equations serve as an intermediate representation (IR)

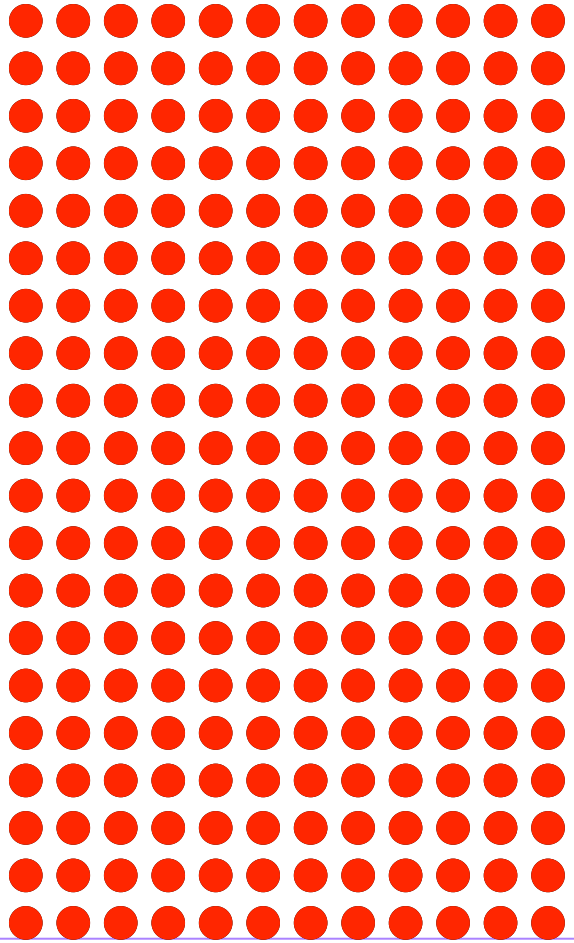


Example: Sequential loop with tiling

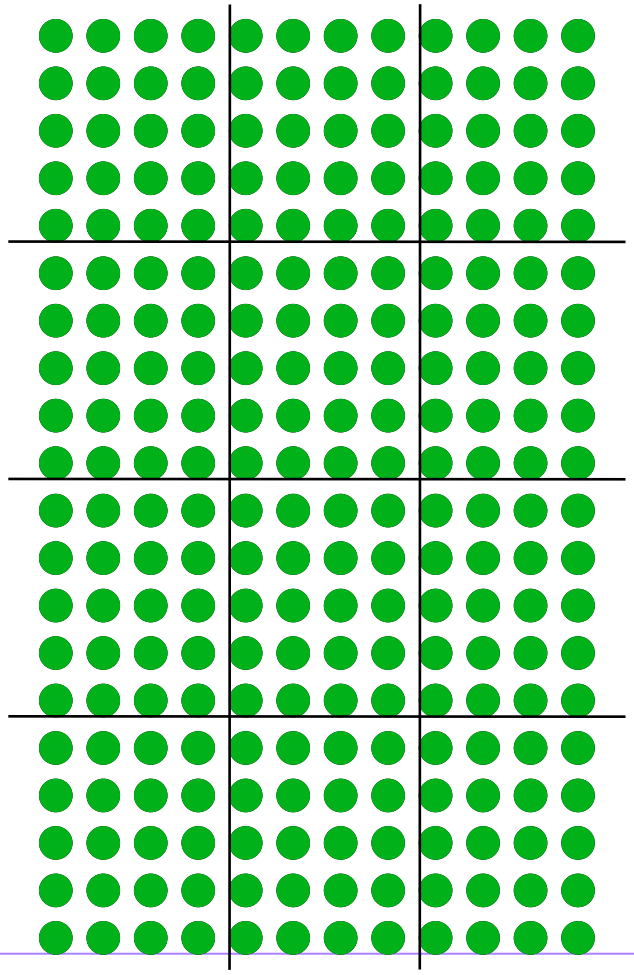
```
for (i=0; i<N; i++)
  for (j=1; j<M; j++)
    A[j] = foo(A[j],A[j-1]);
```

```
for (ti=0; ti<N; ti+=bi
  for (tj=1; tj<M; tj+=bj)
    for (i=ti; i<min(ti+bi,N); i++)
      for (j=tj; j<min(tj+bj,M); j++)
        new-body(i,j);
```

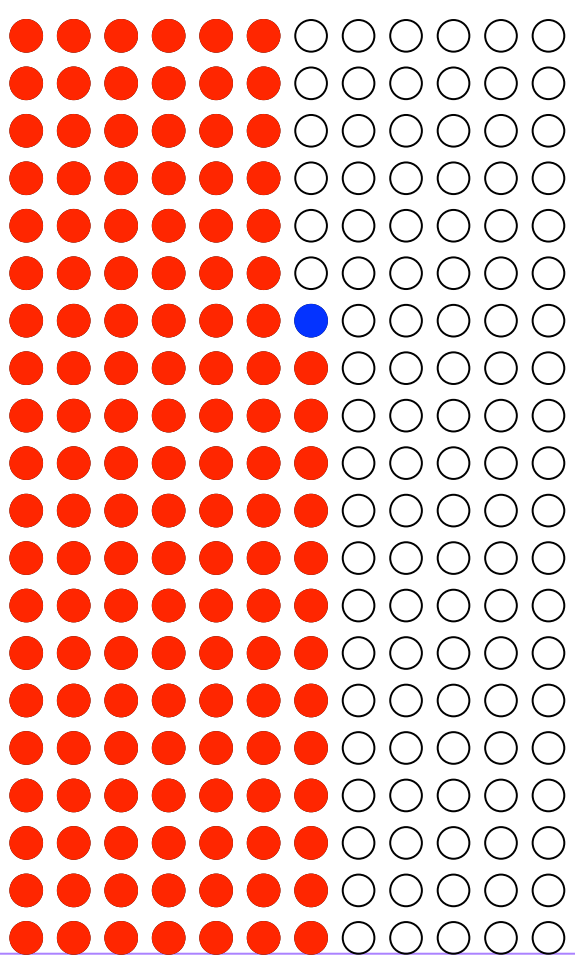
Sequential execution order

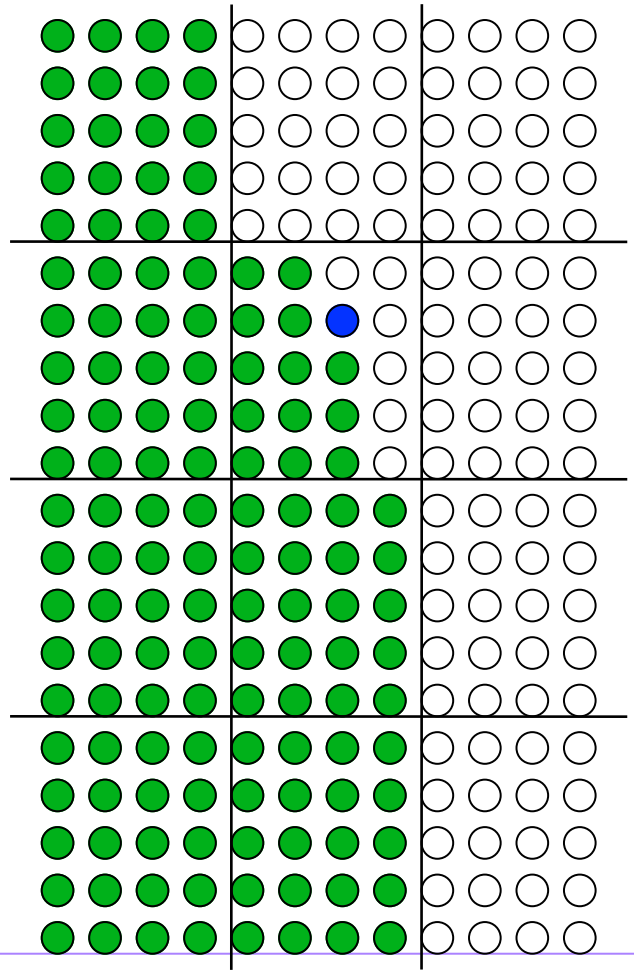


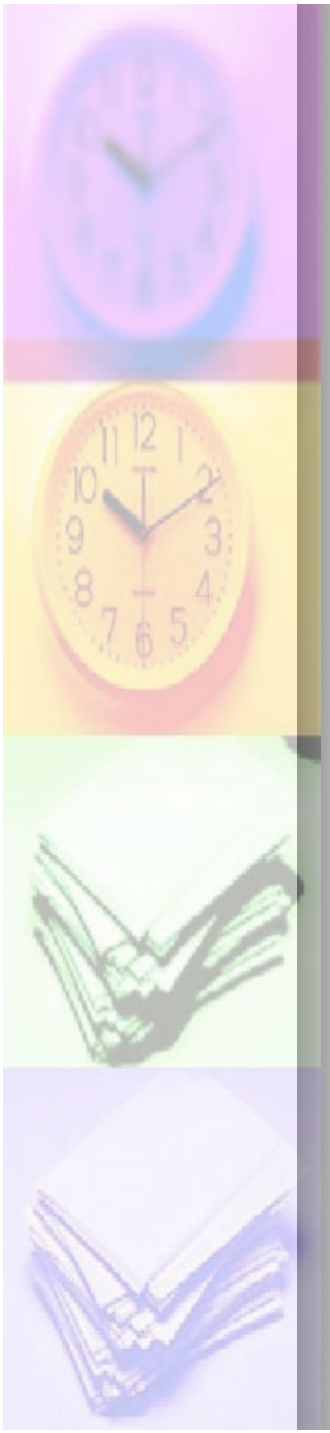
With 4x5 tiles (still sequential)



At iteration ⟨i,j⟩ what values are live?





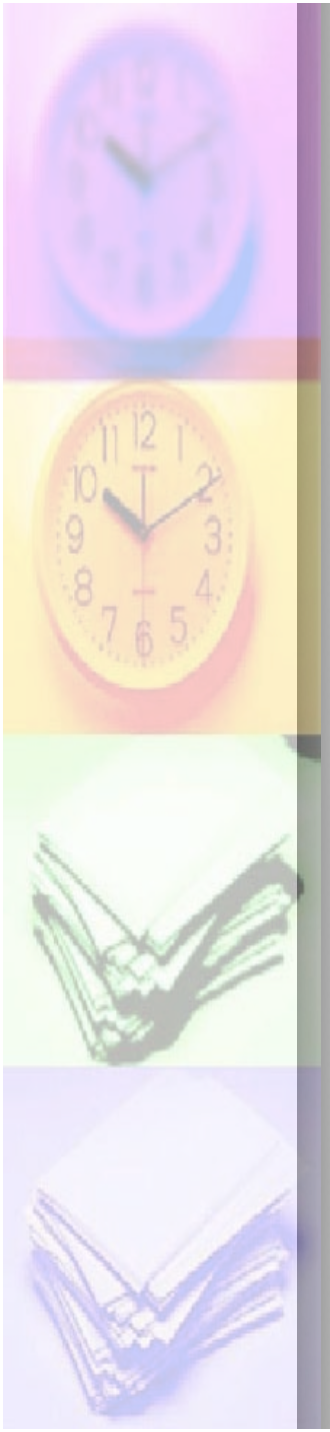


So, what is the new body?

- A legal memory allocation function:

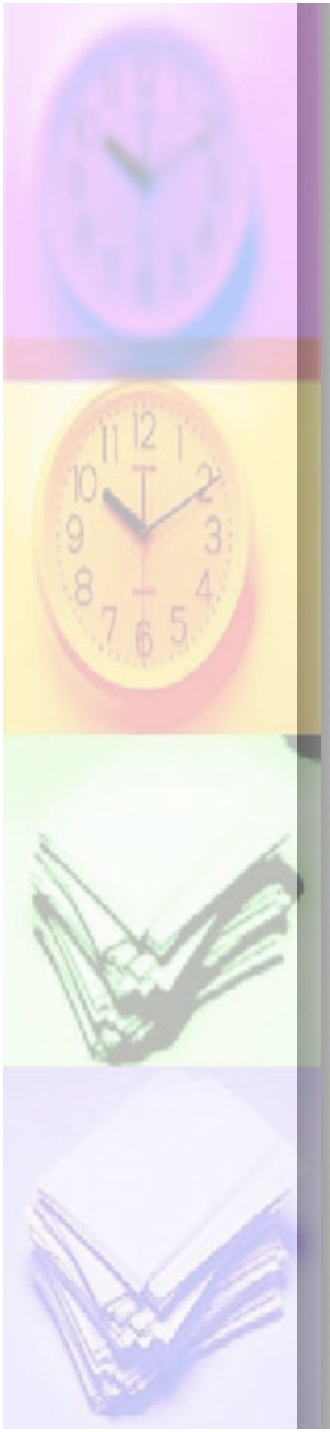
$$(i - j) \bmod (N + b_i)$$

```
#define MEM(i,j) (i-j) mod (M+bi);
for (ti=0; ti<N; ti+=2)
  for (tj=1; tj<M; tj+=2)
    for (i=ti; i<min(ti+1,N); i++)
      for (j=tj; j<min(tj+1,M); j++)
        A[MEM(i,j)]= foo(A[MEM(i-1,j)], A[MEM(i,j-1)]);
```



What is the new body?

- Tiling necessitates revisiting the memory allocation of the original program
- Needs lifetime analysis
- Equational form makes this clearer.



What is the Equation?

- In the original code,
  - at the moment when iteration ⟨i,j⟩ is executed
  - two values are read, passed as arguments to `foo`, the returned value is stored
- Which instance of which iteration produced each of the two values read: `A[j]` and `A[j-1]`
- Analyze the data-flow in the program