Application of Fortran Pthreads on Linear Algebra and Scientific Computing

Clay P. Breshears

Henry A. Gabb

Mark Fahey

USAE Waterways Experiment Station

Major Shared Resource Center

Acknowledgements

This work was funded by the DoD High Performance Computing Modernization Program CEWES Major Shared Resource Center through Programming Environment and Training (PET), Contract Number: DAHC 94-96-C0002, Nichols Research Corporation.

Introduction

- Last year: Introduced F90 Pthreads API
- This year: Are they really useful?
  - How easy is programming?
  - Can we get decent parallel performance?
  - Are there algorithmic considerations?
  - Are there external considerations?
- Must be within a user's attention span

Pthreads

- POSIX standard for thread functions
  - Thread management
  - Mutual exclusion
  - Conditional variables
  - Attributes

- Only defined in C

F90 Pthreads API

- F90 subroutines interface C functions
- ‘f’ prefix to name
  - fpthread_create
  - Similar to PVM, MPI
  - Error code as final argument
- F90 module and C wrapper routines

Problems Considered

- Matrix Multiplication
- Direct Solution of Linear Systems
  - Gaussian Elimination and Back Substitution
- Command, Control, Communication, and Intelligence (C3I) Benchmarks
  - Map-Image Correlation
  - Terrain Masking

Project Goals

- Apply threaded programming techniques to scientific computations
- Demonstrate and exploit concurrency in numeric codes
- Achieve execution speed up with multiple threads/processors

NOT a Project Goal…

To produce the fastest executing versions of codes and algorithms examined

Our Emphasis:

- How easy is the method to use?
- How much speed up might be expected?

Matrix Multiplication

- Sparse matrix-matrix multiplication
- Row-major linked list data structure
  - Direct Methods for Sparse Matrices by Duff, Erisman and Reid
- IKJ loop structure
  - C(I, :) = C(I, :) + A(I, K) * B(K, :)
  - I^{th} row of C; K^{th} row of B
  - Scalar from A (accessed across I^{th} row)

Thread Algorithm

- While more rows to process
  - get next row number (lock shared counter)
  - for each element in row of A do
    - copy appropriate row of B to local vector
    - multiply vector by scalar from A
    - add results to local “summation” vector
  - add “summation” vector to row of C
    - lock data structure to prevent overwriting

1000x1000 Sparse Matrix Multiplication (< 10K NZ) on SGI/Cray Origin 2000 (IRIX 6.4)

# of Threads	Time (seconds)
1	0.259
2	0.261
4	0.261
8	0.264
16	0.270
32	0.301
64	0.324
128	0.364

Dense Matrices

- Not enough work in sparse case
- IKJ loop structure
  - row-major access
- JKI loop structure
  - C(:, J) = C(:, J) + A(:, K) * B(K, J)
  - J^{th} column of C; K^{th} column of A
  - Scalar from B (accessed down J^{th} column)

1000x1000 Dense Matrix Multiplication on SGI/Cray Origin 2000 (IRIX 6.4)

# of Threads	IKJ Time (seconds)	JKI Time (seconds)
1	75.5	29.8
2	42.4	20.2
4	22.4	11.2
8	11.9	6.1
16	6.3	3.4
32	3.6	2.0
64	3.2	1.9
128	3.0	2.3

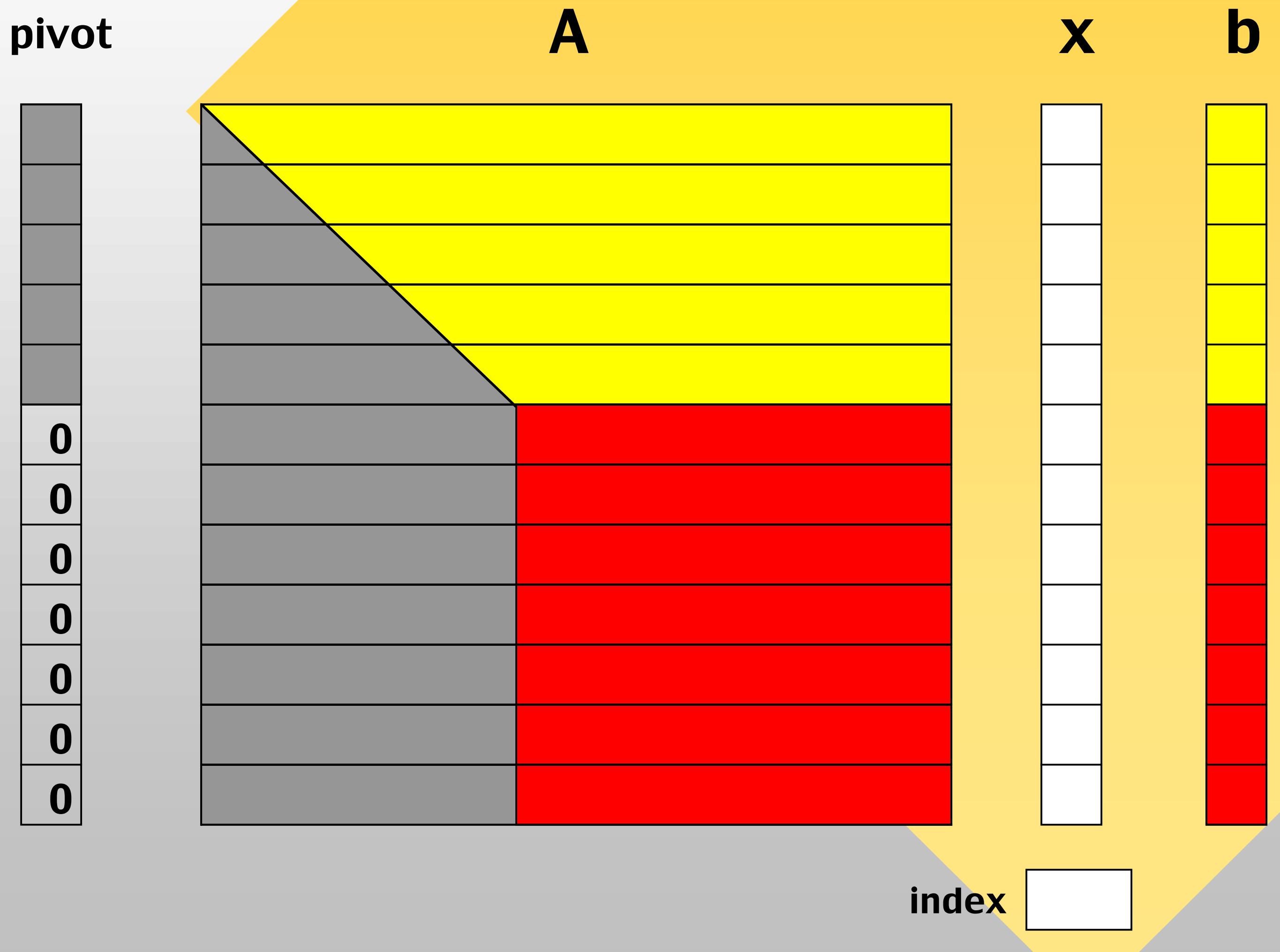
Solution of Linear Systems

- Gaussian Elimination with Back Substitution
  - simple method with row updates
  - diminishing amounts of work

Q: How do we distribute work evenly among threads throughout computation?

A: Cyclic distribution of rows

Cyclic Row Partitioning

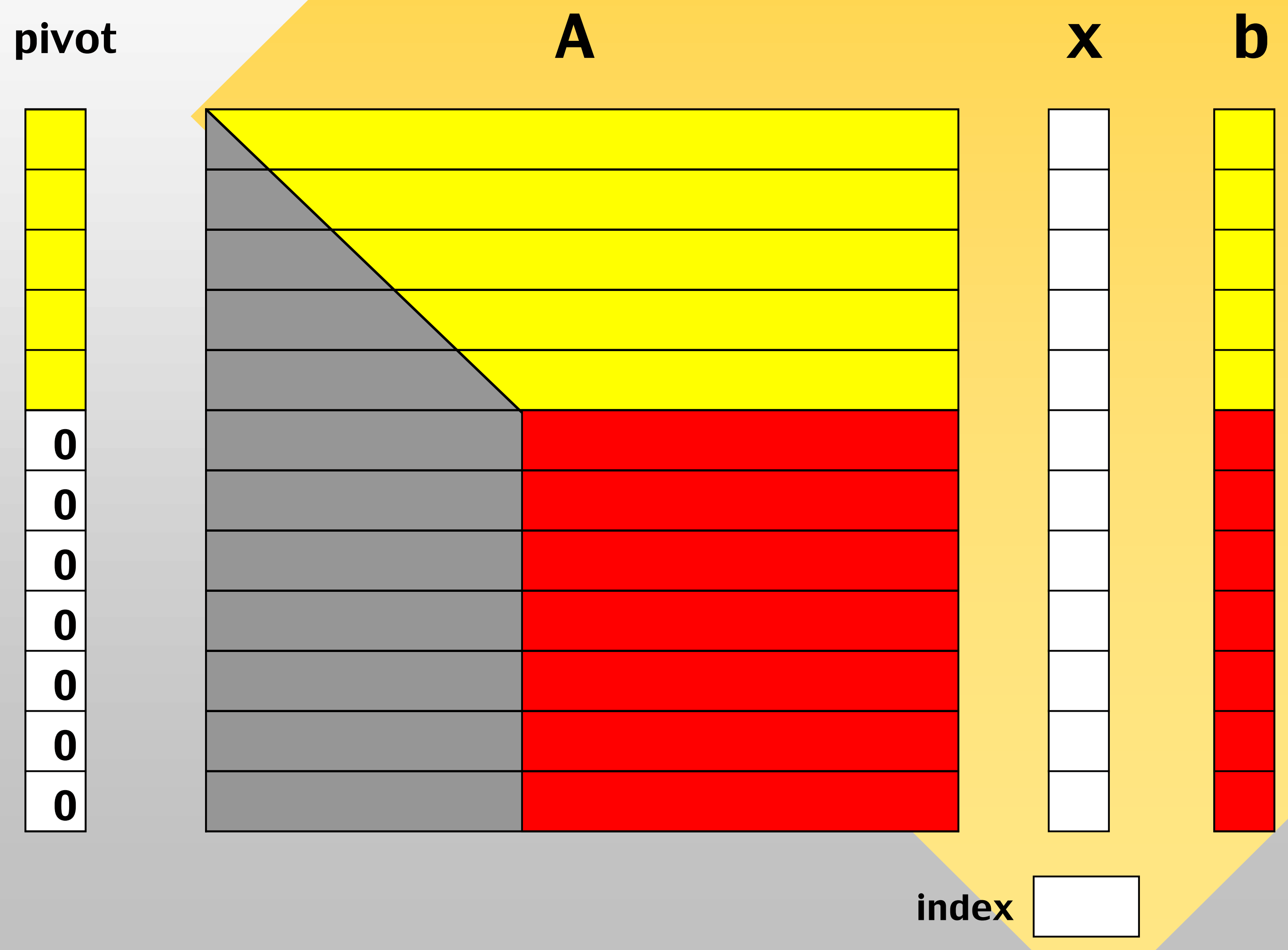


Thread Algorithm

```
do i = 1, NUMROWS
   if (i mod myid = = 0) then
       save pivot, row(i)
   else
      wait for pivot to be saved
   endif
   copy row(i), pivot to local row,
pivot
   do j = i+1, NUMROWS
      if (j mod myid = = 0) then
         compute row multiplier
         update row(j) with local row
      endif
   end do
end do
```

```
indx = NUMROWS
while (indx > 0)
   while (indx mod myid = = 0)
      fpthread_cond_wait
   compute x(indx)
   indx = indx - 1
   fpthread_cond_broadcast
end while
```

Cyclic Row Partitioning



But What About…

- Column-Major ordering is Fortran standard
  - Helped with Matrix Multiply code

- Modify threaded algorithm
  - Transpose A matrix after input
    - library routine is threaded
  - Swap A(i,j) indices for A(j,i)

Timing Results

F90 threaded Gaussian Elimination with back substitution
2000x2000 system of equations on SGI/Cray Origin2000

# of Threads	Row-Major	Transpose Column-Major
1	672	162
2	735	232
4	556	115
8	1030	100
16	1642	147
32	1667	131

But What About…

- …Memory contention of threads if matrices stored on a single processor?
  - Used _DSM_ROUND_ROBIN to no significant effect
  - Used A(CYCLIC,*) distribution to no significant effect
- …Distribution of threads to processors?

pthread_setconcurrency

- SGI extension to Pthreads
- Wrote F90 wrapper
- Set to number of threads executing

Added Timing Results

F90 threaded Gaussian Elimination with back substitution
2000x2000 system of equations on SGI/Cray Origin2000

# of Threads	Row-Major	Transpose Column-Major	Transpose fp_setconcurrency
1	672	162	
2	735	232	125
4	556	115	68
8	1030	100	26
16	1642	147	53
32	1667	131	64

10K System of Equations

- Transpose algorithm
  - 32 threads
  - 10144 seconds on Origin2000
- Transpose with setconcurrency
  - 32 threads
  - 8608 seconds on Origin2000

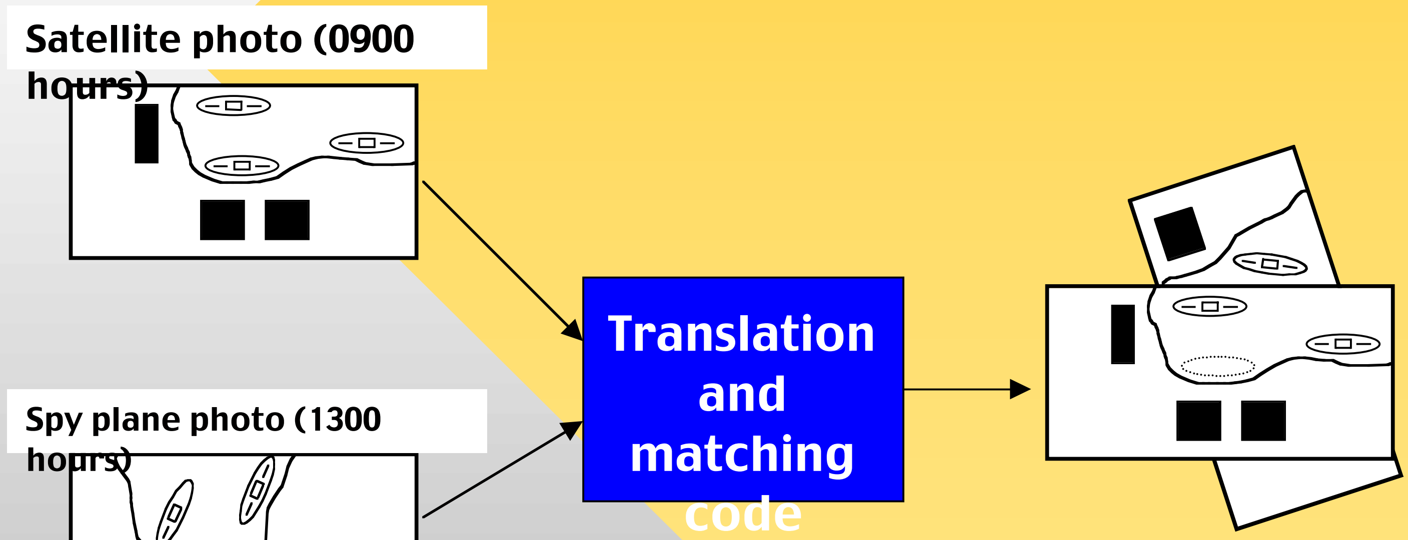
C3I Benchmarks

- U.S. AFRL Information Directorate
  - Rome Research Site (Griffis AFB)
- 10 non real-time C3I functions
  - diverse
  - computationally
  - challenging
  - representative of C3I systems
- Spec, sequential code, associated dataset

Map-Image Correlation

- Surveillance data from remote sensors
  - space-based infrared satellites
  - remotely-piloted vehicles
  - intelligence photographs
- Determine the alignment of features in the images with a detailed map of the area
- Potential for comparing “before” and “after” images

Example Problem



Find matching rotation of physical features and realize a ship has departed

Potentials for Concurrency

- Each image is independent
- 2-D Fast Fourier Transform
  - Each column is independent 1-D FFT
  - Transpose
  - Each column (original row) is independent
- FFT of finite number of rotations is independent

Thread Algorithm

Create thread for each image (both original and rotations):

- Discretize image
- For each column in image create thread for 1-D FFT
- Transpose image array
- For each column in image create thread for 1-D FFT

Join threads

Correlate images (using threaded Inverse Fourier Transform)

Implementation Details

- Two 1024x1024 Grids
- Use 1-D FFT routine from Cray Sci Lib
- Total of three 2-D FFTs
  - Two images
  - Inverse FFT for correlation
- Use pthread_setconcurrency

Map-Image Timing Results

Threaded Image Correlation of Two 1024x1024 grids on SGI Origin2000

Number of threads	1	2	4	8	16	32	64
Time (seconds)	155	85	47	28	19	14	16

Single Threaded 2-D FFT (Cray Sci Library): < 2 seconds

Terrain Masking

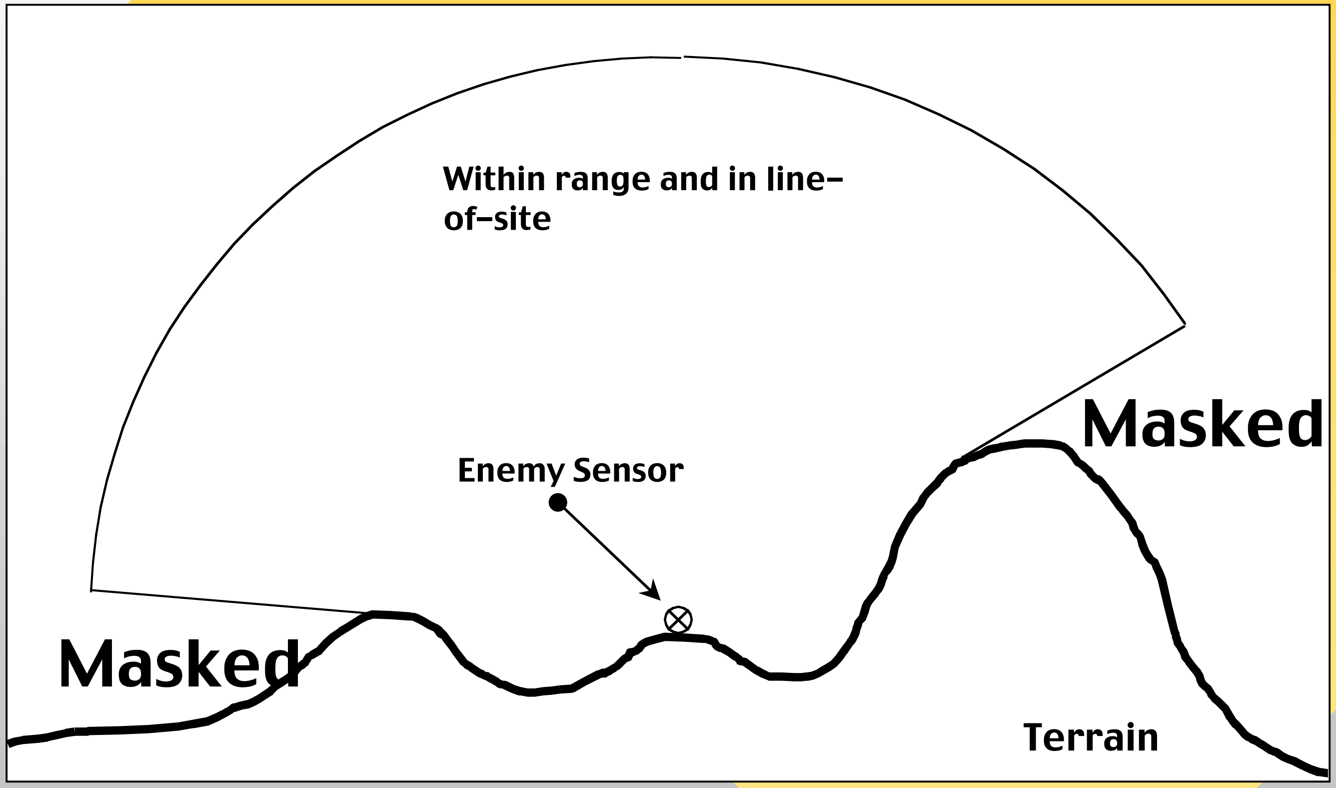
- Used in aircraft flight mission computer systems to aid in attack, covert, and evasive flight operations
- Compute evasive routes with low observability
  - given a set of threats and their positions

Problem Description

- Input:
  - 2-D relief map (grid of surface elevations)
  - Position and range of *threats* within region

- Output:
  - Original map plus masking altitudes
    - minimum visible altitude at grid points

Threat Range and Line of Sight

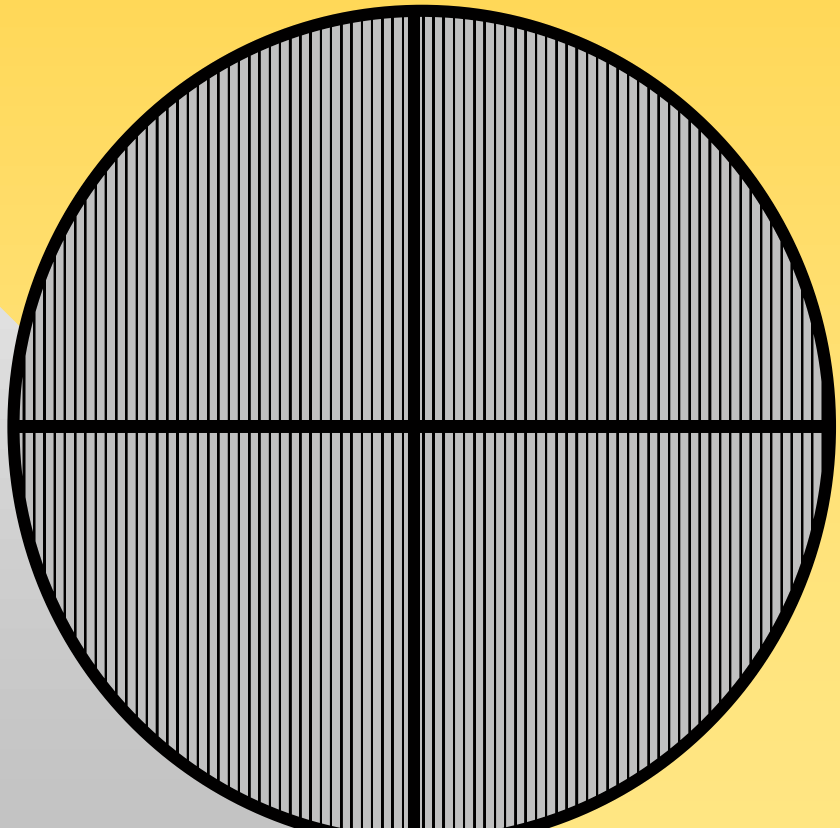


Concurrent Method 1

For each threat

- determine range boundaries of threat
- divide range into N sectors
- create N threads
    - assign one per sector
- join threads

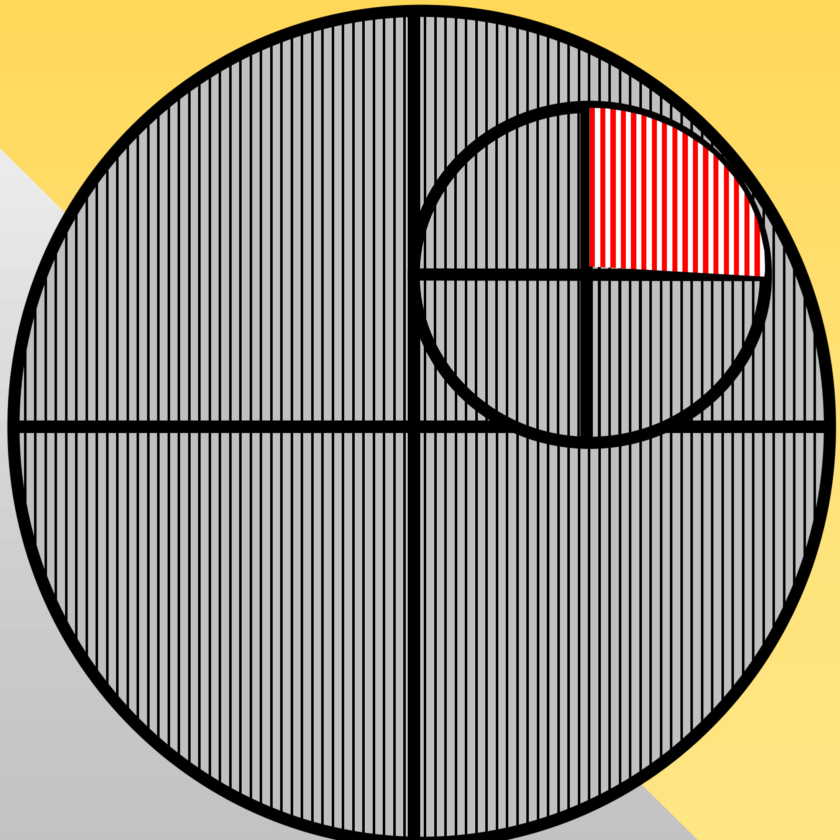
Sector Concurrency Model



Concurrent Method 2

```
Determine number of sectors, N
create M (number of threats) threads
do I = 1, N
   for each threat compute sector I
         lock potential overlap areas
   synchronize M threads
end do
```

Threat Concurrency Model



Create and use mutex for each overlapping column

Terrain Masking Results

Terrain Masking Timing (in minutes) for 6000x6000 element grid with 90 threats on SGI Origin2000

NUMBER OF THREADS	1	2	4	8	16
Time (minutes)	25	14	8	5.5	3.5

With or without pthread_setconcurrency yielded little difference.

Conclusions

- Threaded codes can demonstrate speed-up
- Minor coding changes to add Pthreads
  - task parallelism or functional units
- Able to take advantage of nested parallelism
- Must still be aware of architectural quirks
  - cache access patterns
  - data distribution and memory contention
  - thread to processor mapping