Vectorization

Lecture 3

Swapnil Singh
LB, KTU

November 27, 2024

Fundamental Concepts

- Definition: Replacing loop-based operations with array operations
- Core principle: Leveraging MATLAB's optimized array operations
- Performance implications: Reduced overhead and improved execution speed
- Memory considerations: Efficient utilization of computational resources

Vectorization Principles

Key Advantages:

- Elimination of explicit loops
- Improved code readability
- Enhanced execution efficiency
- Better memory management

Primary Applications:

- Numerical computations
- Data preprocessing
- Signal processing
- Statistical analysis

Element-wise Operations

Traditional Approach vs. Vectorized Solution

```
% Non-vectorized
for i = 1:length(x)
  y(i) = sin(x(i))^2 + cos(x(i))^2;
end

% Vectorized
y = sin(x).^2 + cos(x).^2;
```

- Note the use of element-wise operators (.)
- Simplified syntax and improved performance
- Automatic parallel execution capabilities

Efficient Matrix Computations

```
% Computing pairwise distances
% Non-vectorized
for i = 1:size(X,1)
  for j = 1:size(X,1)
      D(i,j) = norm(X(i,:)-X(j,:));
  end
end

% Vectorized
D = sqrt(sum((X - permute(X,[3,2,1])).^2,2));
```

- Exploitation of MATLAB's matrix operations
- Dimensional analysis for optimal implementation

Logical Indexing

Advanced Selection Techniques

```
% Find and replace values
% Non-vectorized
for i = 1:length(x)
  if x(i) < threshold
      x(i) = 0;
  end
end

% Vectorized
x(x < threshold) = 0;
```

- Boolean array operations
- Conditional vector assignments
- Mask-based computations

Broadcasting and Expansion

Implicit Dimension Handling

```
% Broadcasting example
A = rand(100,1);
B = rand(1,100);
C = A + B;  % 100x100
```

Key Concepts:

- Automatic size matching
- Dimension compatibility
- Memory efficiency

Performance Analysis

Comparative Benchmarking

- Execution time measurements

```
tic; % vectorized code; toc
```

- Memory profiling techniques

```
profile on; % code; profile viewer
```

- Optimization strategies
    - Preallocating arrays
    - Avoiding temporary arrays
    - Utilizing built-in functions

Common Pitfalls

Optimization Challenges

- Excessive memory allocation in vectorized operations
- Inappropriate use of cell arrays in numerical computations
- Inefficient handling of sparse matrices
- Suboptimal implementation of conditional operations

```
% Memory-intensive operation
result = arrayfun(@heavy_function, ...
                  large_array);  % Avoid
```

Best Practices

Guidelines for Efficient Vectorization

Do:

- Preallocate arrays
- Use built-in functions
- Profile code performance
- Consider memory usage

Avoid:

- Growing arrays incrementally
- Unnecessary type conversions
- Complex nested loops
- Redundant computations

Advanced Applications

Real-world Implementation Scenarios

- Image processing operations
- Signal analysis algorithms
- Statistical computations
- Machine learning implementations
- Numerical optimization routines

```
% Example: Image filtering
filtered = conv2(image, kernel, 'same');
```

Summary

Key Takeaways

- Vectorization as a fundamental optimization strategy
- Balance between code readability and performance
- Importance of proper implementation techniques
- Consideration of hardware limitations
- Role in modern scientific computing