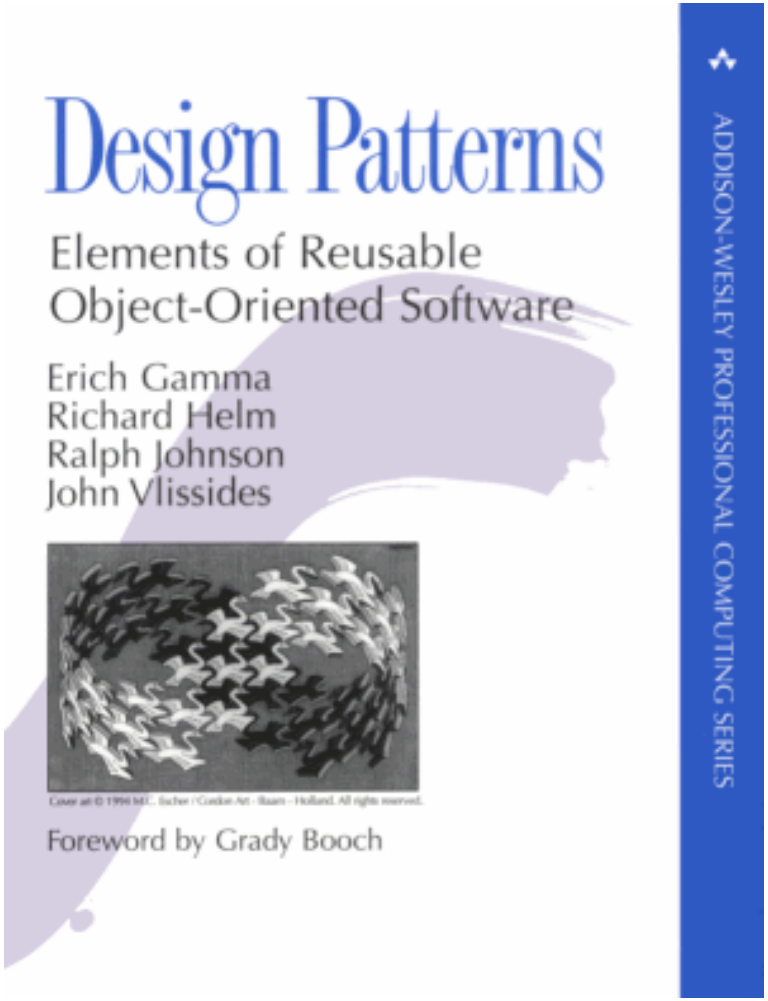
(A project to develop) A Pattern Language for Parallel Programming

Ralph Johnson

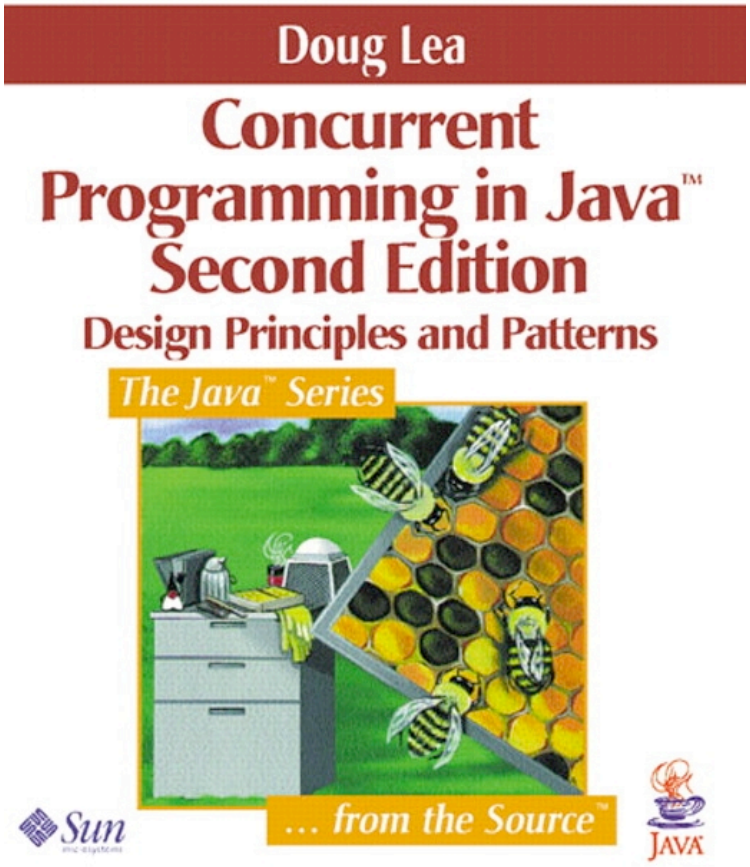
University of Illinois at Urbana-Champaign

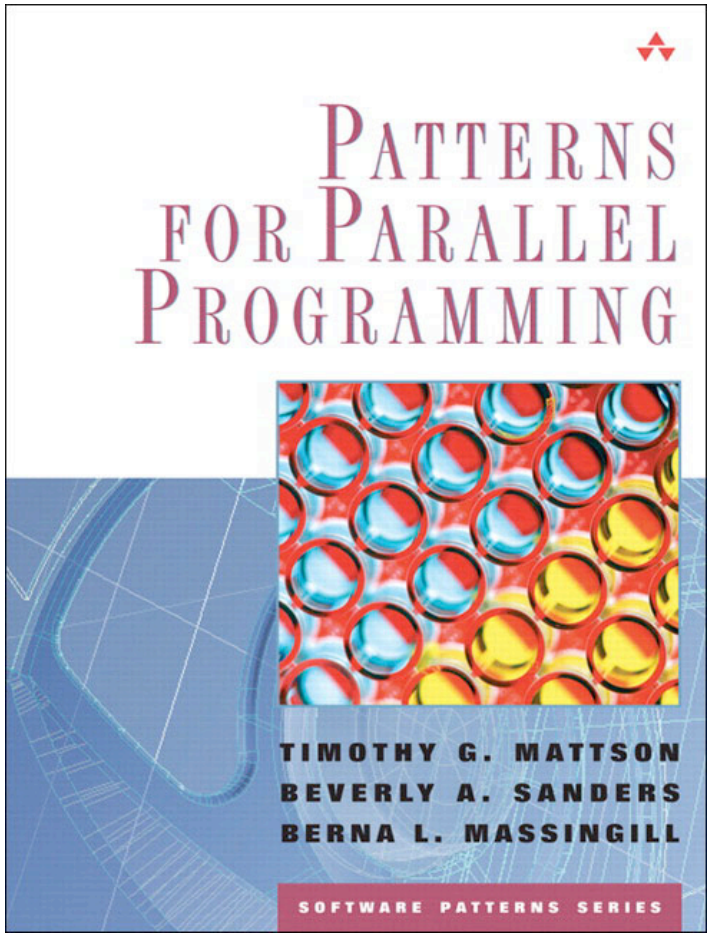
rjohnson@illinois.edu


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Pattern language

- Set of patterns that an expert (or a community) uses
- Patterns are related (high level-low level)


Doug Lea
Concurrent Programming in Java™ Second Edition
Design Principles and Patterns
The Java™ Series
... from the Source™
Sun
JAVA


PATTERNS
FOR PARALLEL
PROGRAMMING
TIMOTHY G. MATTSON
BEVERLY A. SANDERS
BERNA L. MASSINGILL
SOFTWARE PATTERNS SERIES

Making a pattern language for parallelism is hard

- Parallel programming
  - comes in many styles
  - changes algorithms
  - is about performance

Our Pattern Language

- Universal Parallel Computing Research Center
- Making client applications (desktop, laptop, handheld) faster by using multicores

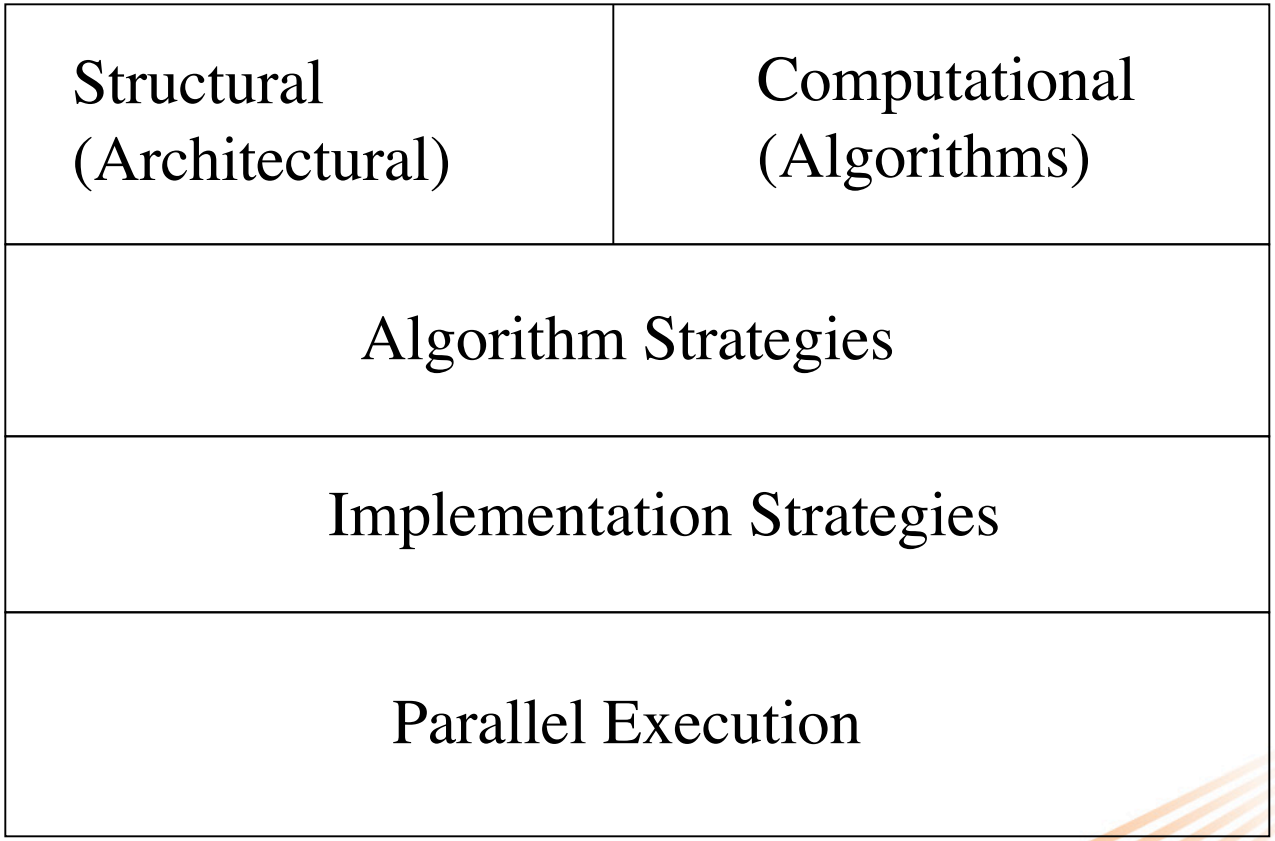
- Kurt Keutzer - Berkeley
- Tim Mattson - Intel

- http://parlab.eecs.berkeley.edu/wiki/patterns
- Comments to rjohnson@illinois.edu

The problem

- Multicores (free ride is over)
- Caches
- Vector processing

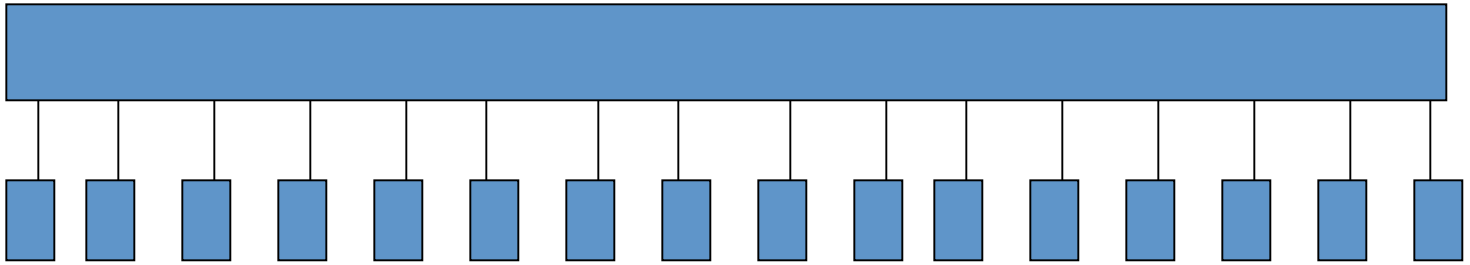
Our Pattern Language



Algorithm Strategies

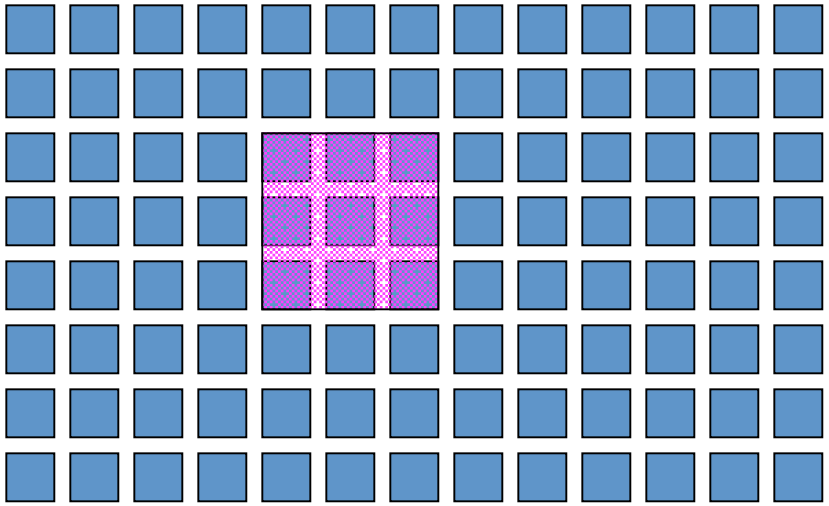
- Task parallelism
- Geometric decomposition
- Recursive splitting
- Pipelining

Task Parallelism



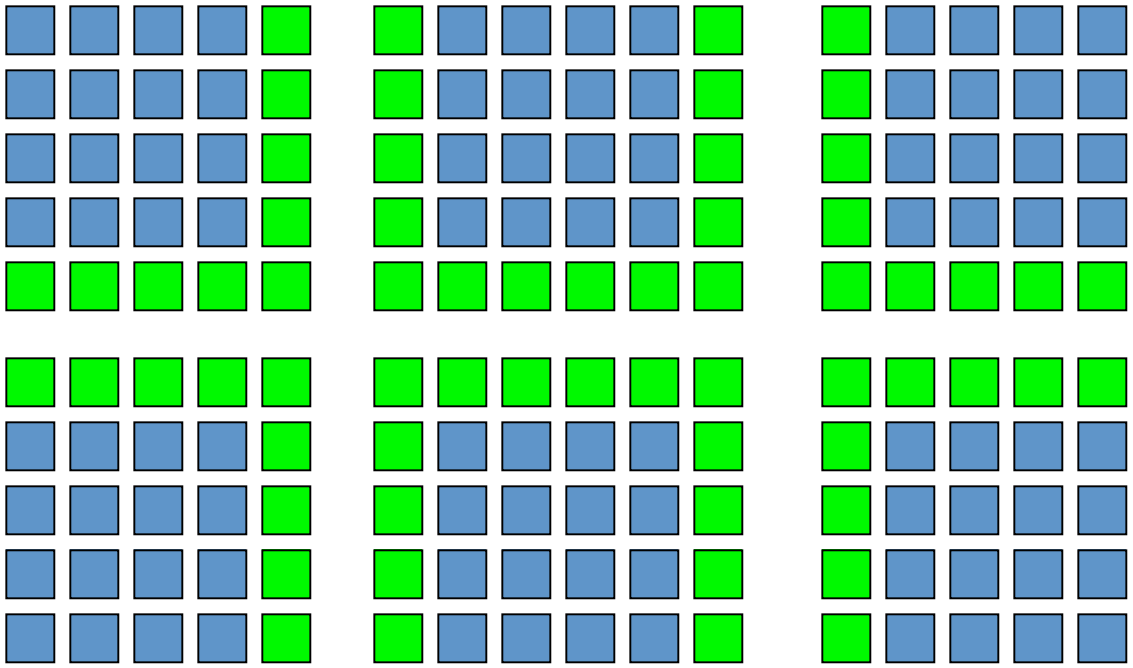
- Communication? As little as possible.
- Task size? Not too big, not too small.
- Scheduling? Keep neighbors on same core.

Geometric Decomposition



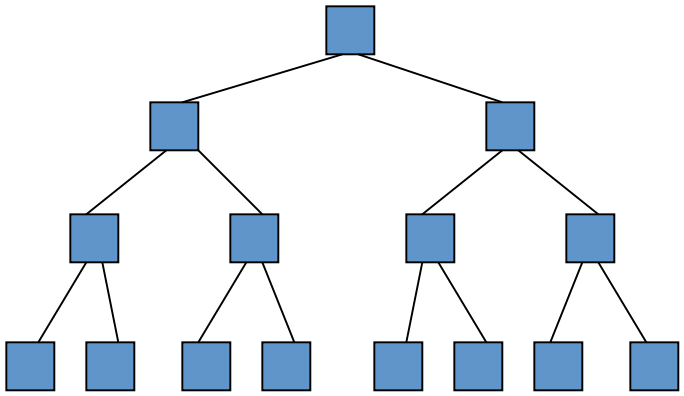
- Stencil

Geometric Decomposition



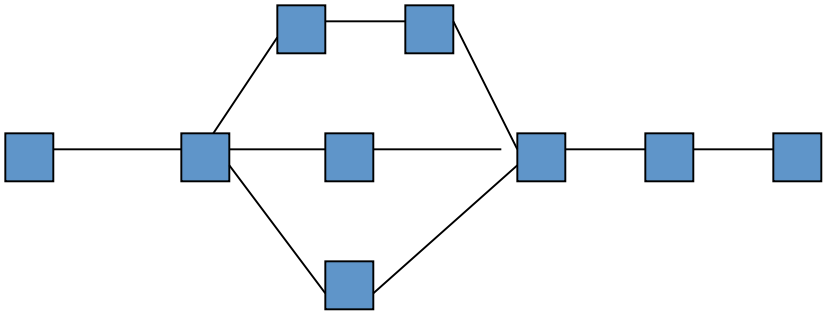
- Ghost cells

Recursive Splitting



- How small to split?

Pipelining



- Bottleneck
- Throughput vs. response time

Parallel Programming Style

- SIMD - data parallism
  - Deterministic semantics, operations on collections
- Fork-join tasking - shared memory
  - Hopefully deterministic semantics, no I/O
- Actors - asynchronous message passing - no shared memory
  - Nondeterministic, good for I/O

Fork-join Tasks

- Tasks are objects with behavior “execute”
- Each thread has a queue of tasks
- Tasks run to completion unless they wait for others to complete
- No I/O. No locks.

```
void tracerays(Scene *world) {
    for (size_t  i = 0, i>WIDTH, i++) {
        for (size_t  j = 0,  j>HEIGHT, j++) {
            output[i][j] = traceray(i,j,world);
        }
    }
}
```

```
#include “tbb/parallel_for.h”
#include “tbb/blocked_range2d.h”
using namespace tbb;

class TraceRays {
    Scene *my_world;
Public:
    void operator() (const blocked_range2d<size_t>& r) {
    …
    }
    TraceRays(Scene *world) {
        my_world = world;
    }
}
```

```
void operator() (const blocked_range2d<size_t>& r) {
    for (size_t  i = r.rows().begin(), i != r.rows().end(), i++) {
      for (size_t  j = j.cols().begin(),  j!=r.cols().end(), j++) {
            output[i][j] = traceray(i,j,world);
      }
    }
}

void tracerows(Scene *world) {
parallel_for(blocked_range2d<size_t>(0,WIDTH,8,0,HEIGHT,8),
    TraceRays(world);
}
```

- Parallel reduction
- Lock-free atomic types
- Locks (sigh!)

- TBB: threadedbuildingblocks.org
- Java concurrency: http://g.oswego.edu/
- Microsoft TPL and PPL: http://msdn.microsoft.com/concurrency

- http://parlab.eecs.berkeley.edu/wiki/patterns
- Comments to rjohnson@illinois.edu