Module 3 Part 4
Application Design for Manycore Software Patterns

Carnegie Mellon University
18-645

Jike Chong
Ian Lane

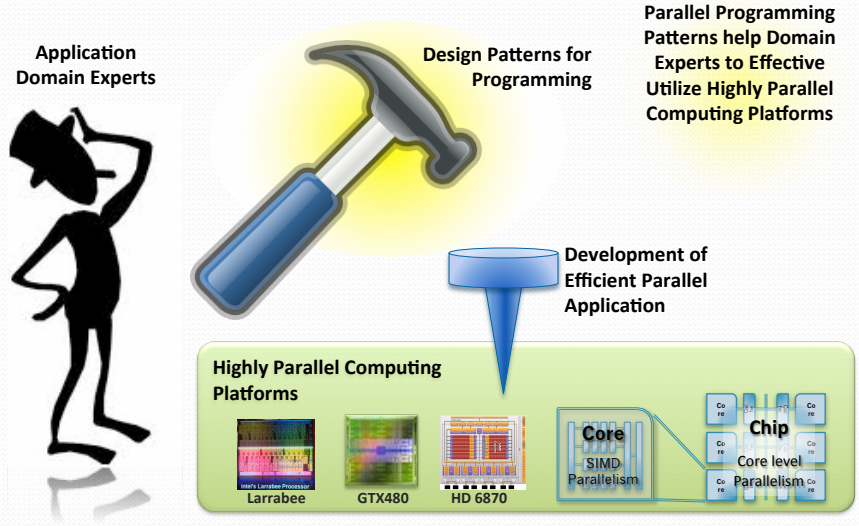
Answers you should know after this…

- What are parallel software patterns?
- What are the three goals the software patterns aim to achieve?
- What is a software architecture?
- How is it important for writing fast code?
- What are the five categories of patterns in OPL?
- What are the nine sections in an OPL pattern?
- What are the areas of consideration for your Term Project?

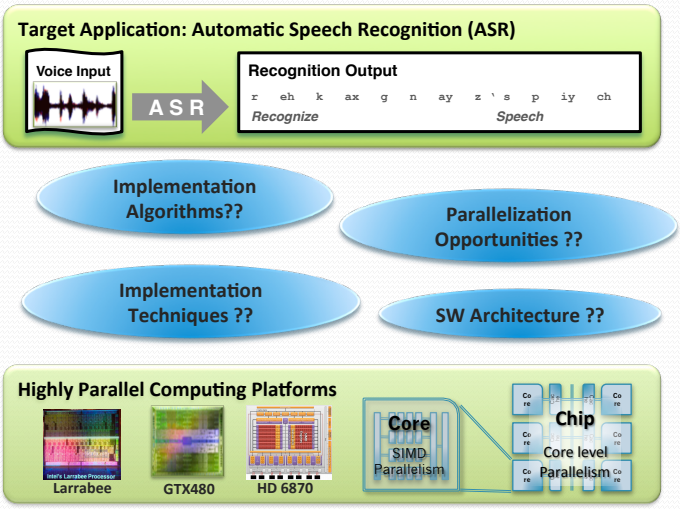
Outline

- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern
- Applications to Your Term Projects

Guide to Design Patterns



Many Difficult Design Decisions



Parallel Software Patterns

A parallel software pattern is a **generalizable solution** to a class of **recurring problems** that occurs in the design of parallel software.

- Attaches names to ***well-analyzed solutions*** that ***encapsulate*** the way ***an expert*** in the field ***solves problems***
- Aims to achieve three goals:
  - Define a set of **vocabularies** to **communicate**
  - Present a set of **expert techniques** for beginners to **learn**
  - Allows experts to more quickly **design** **complex systems**

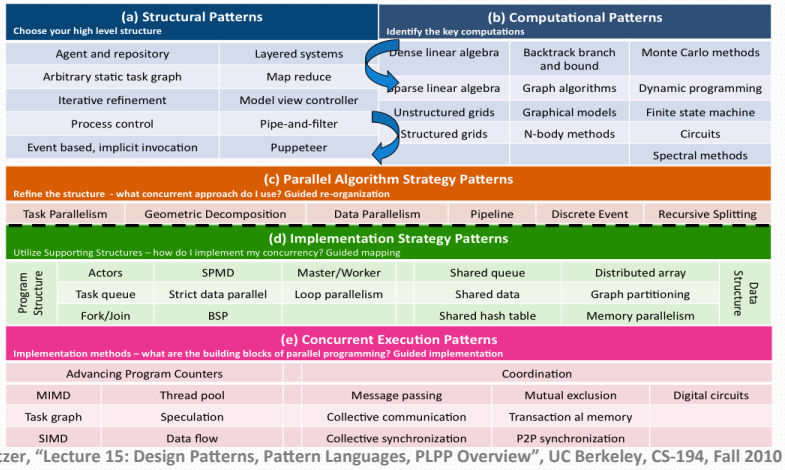
Architecting Parallel Software

- We believe the solution to parallel programming is developing a good software architecture
  - *A **software architecture*** is a hierarchical composition of:
    - Computational patterns – the atoms
    - Structural patterns – the molecular bonds
- This software architecture naturally gives:
  - Modularity
    - Efficient management
    - Efficient implementation
    - Efficient verification
  - And … it identifies key computations, invariants, and interfaces

Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

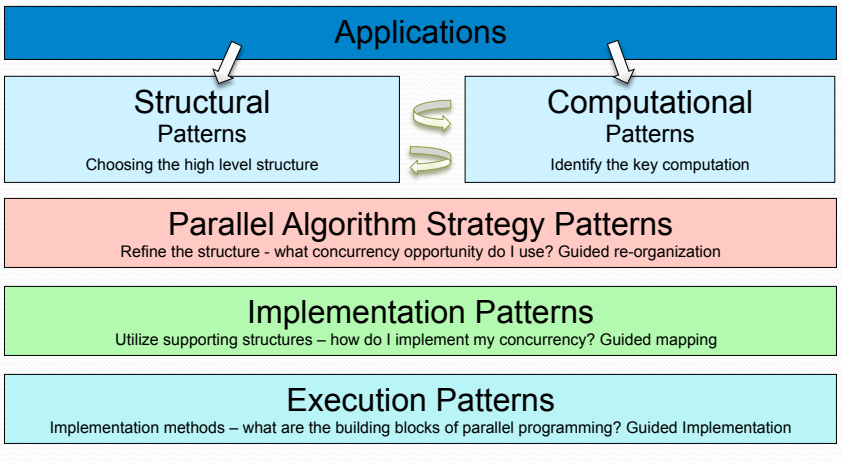
Parallel Software Patterns

- Our Pattern Language: http://parlab.eecs.berkeley.edu/wiki/patterns/patterns



Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

OPL: The Organization



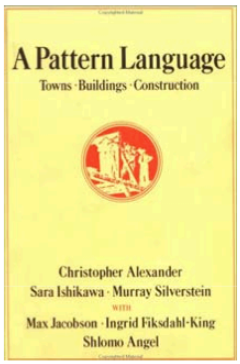
Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Outline

- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern
- Applications to Your Term Projects

Alexander's Pattern Language

- Christopher Alexander's approach to (civil) architecture:
  - "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." *Page x, A Pattern Language,* Christopher Alexander
- Alexander's 253 (civil) architectural patterns range from the creation of cities (2. distribution of towns) to particular building problems (232. roof cap)
- A pattern language is an organized way of tackling an architectural problem using patterns
- Main limitation:
  - It's about civil not software architecture!!!



Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Computational Patterns

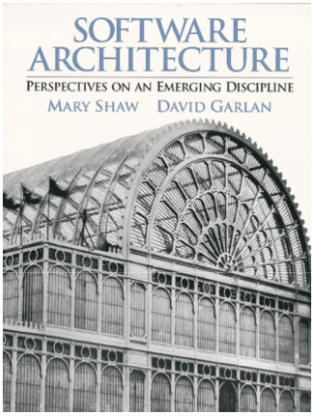
- The computational Dwarfs from "The Berkeley View" (Asanovic st al.)
- Collected to motivate development of parallel computing platforms

Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Unstructured Grid												
Structured Grid												
Dense Matrix												
Sparse Matrix												
Spectral (FFT)												
Monte Carlo												
N-Body												

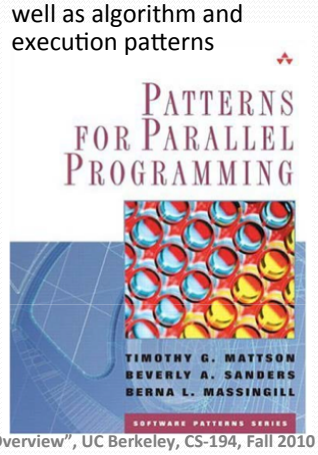
Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Other Important Influences

Inspired many of the Structural Patterns



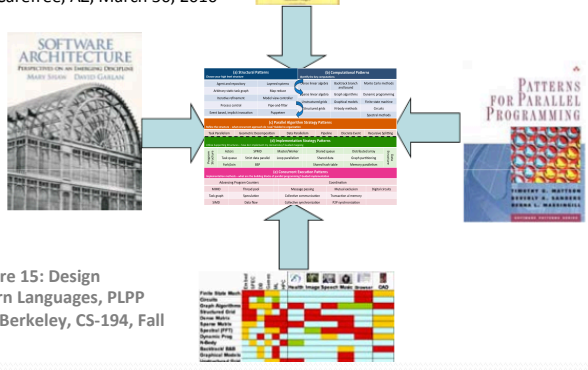
Provides a foundation for the parallelization strategy, as well as algorithm and execution patterns



Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Putting It All Together

Kurt Keutzer, Berna Massingill, Tim Mattson, Beverly Sanders, "**A Design Pattern Language for Engineering (Parallel) Software: Merging the PLPP and OPL projects**", 2nd Annual Conference on Parallel Programming Patterns (ParaPLoP'10), Carefree, AZ, March 30, 2010

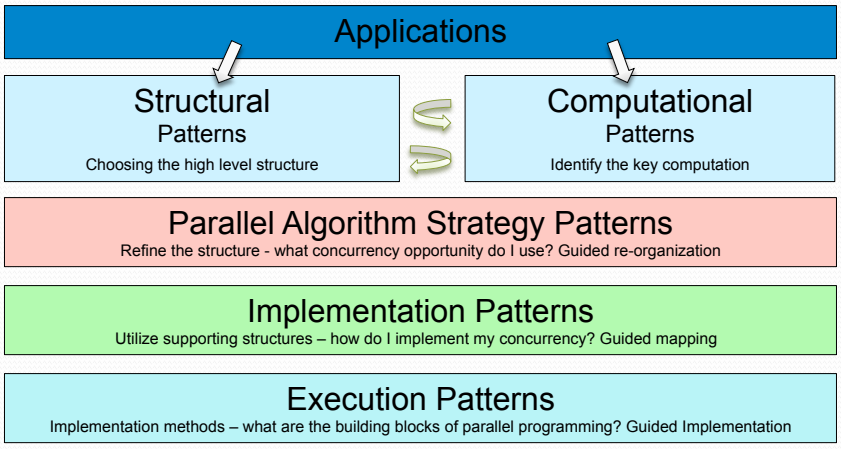


Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Outline

- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern
- Applications to Your Term Projects

OPL: The Organization



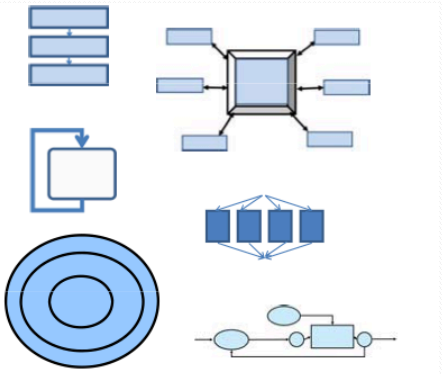
Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Identifying the SW Structure

- Structural Patterns define the structure of the SW but not what is computed

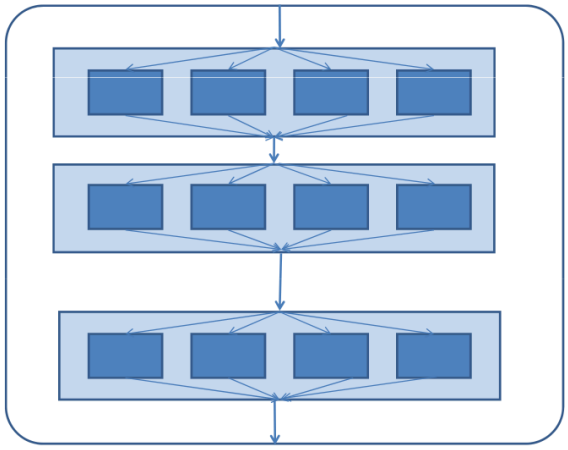
Structural Patterns

- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Layered Systems
- Model-view-controller
- Arbitrary Task Graphs
- Puppeteer
- Iterator/BSP
- MapReduce



Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Analogy: Layout of Factory Plant



Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Identifying Key Computations

- Computational patterns describe the key computations but not how they are implemented

Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Unstructured Grid												
Structured Grid												
Dense Matrix												
Sparse Matrix												
Spectral (FFT)												
Monte Carlo												
N-Body												

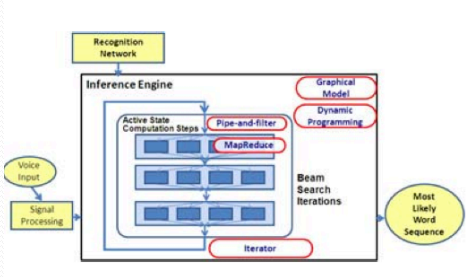
Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Analogy: Machinery of the Factory

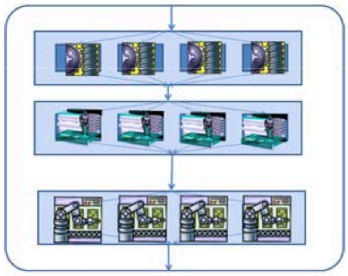
Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Architecting the Whole Application

- Raises appropriate issues like scheduling, latency, throughput, workflow, resource management, capacity, etc



SW Architecture of Large-Vocabulary Continuous Speech Recognition

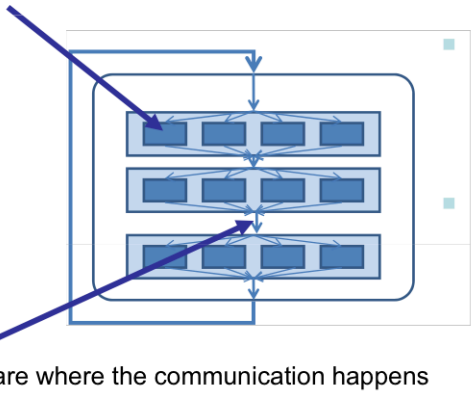


Analogous to the design of an entire manufacturing plant

Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Elements of a Structure Pattern

- Components are where the computation happens



- A configuration is a graph of components (vertices) and connectors (edges)
- A structural patterns may be described as a familiy of graphs.

Connectors are where the communication happens

Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Inventory of Structural Patterns

- **Pipe-and-Filter**
- Agent-and-Repository
- Event-based
- Layered systems
- Model-view-controller
- Arbitrary Task Graphs
- Puppeteer
- Iterator/BSP
- **MapReduce**

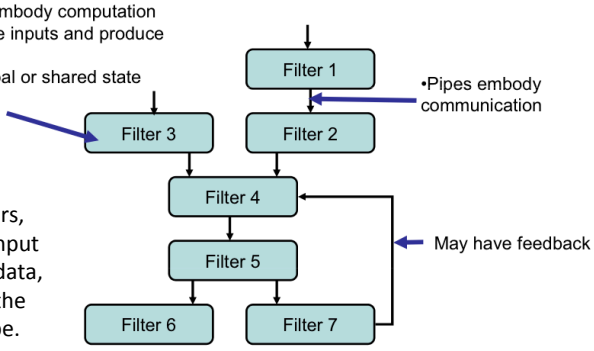
- We build arbitrarily complex software structures out of these patterns

Pattern 1: Pipe and Filter

Problem: Data flowing through modular phases of computation

Solution: Constructs the program as filters (computational elements) connected by pipes (data communication channels).

•Filters embody computation
•Only see inputs and produce outputs
• No global or shared state

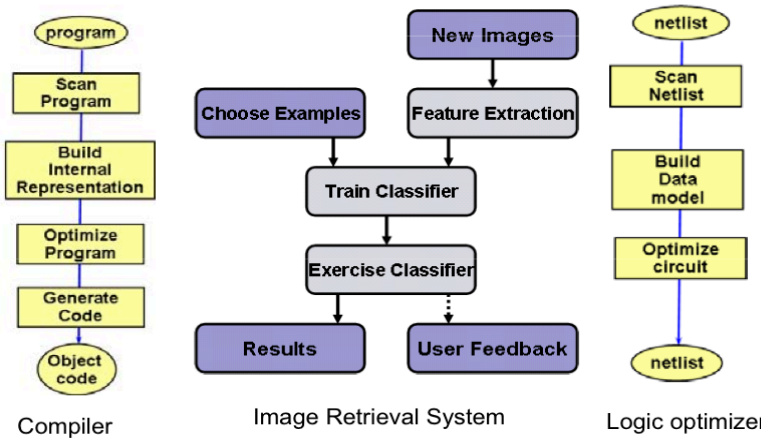


Data flows through the succession of stateless filters, taking input only from its input pipe(s), transforming that data, and passing the output to the next filter via its output pipe.

Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Example of Pipe and Filter

- Almost every large software program has a pipe and filter structure at the highest level



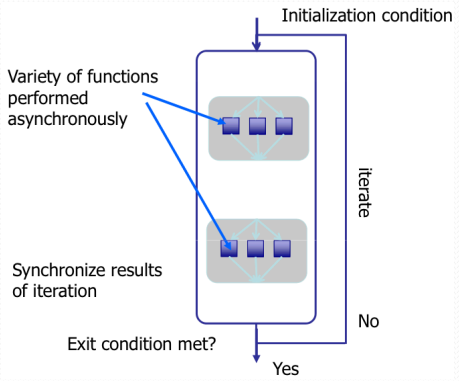
Keutzer, "Lecture 15: Design Patterns, Pattern Languages, PLPP Overview", UC Berkeley, CS-194, Fall 2010

Pattern 2: Iterative Refinement

Problems: The application of a set of operations over and over to a system until a predefined goal is realized or constraint is met

Solution: Wrap a flexible iterative framework around the operation that operates as follows:

- An iteration of the computation is performed
- The results are checked against a termination condition
- Depending on the results of the check
  - the computation completes or proceeds to the next iteration

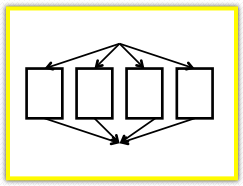


Pattern 3: MapReduce

Problem: Efficiently apply the same function many independent data sets and summarize the results

Solution: Define a program structured as two distinct phases:

- Phase one: a single function is mapped onto independent sets of data
- Phase two: the results of mapping are reduced

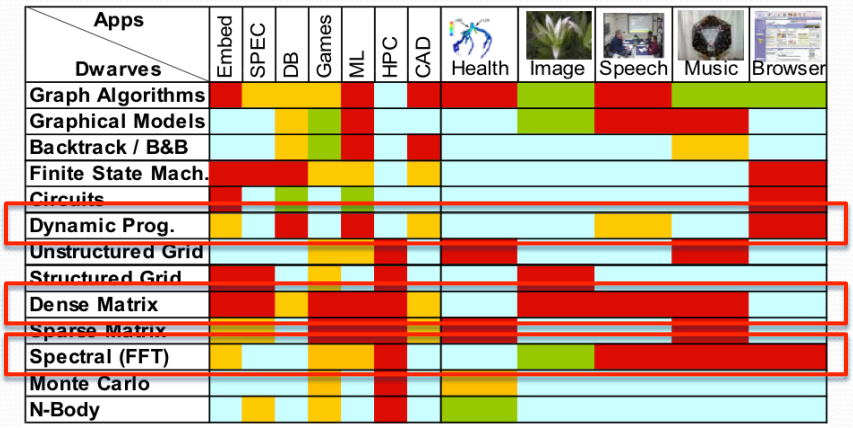


Outline

- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern
- Applications to Your Term Projects

Inventory of Key Computations

- We build arbitrarily complex computations out of these thirteen computational patterns



Pattern 1: Linear Algebra

Problem: Many computation can be expressed in terms of linear operations applied to matrices and vectors for which most elements are non-zero

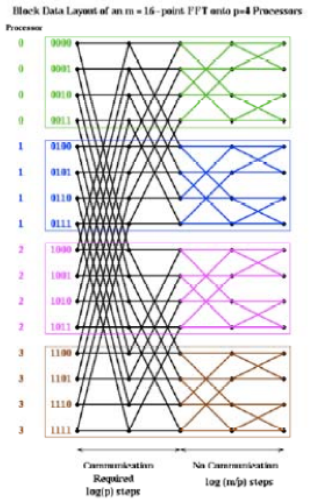
Solution: Organize computation as a sequence of arithmetic expressions acting on dense arrays of data. Often make heavy use of standard library routines called the Basic Linear Algebra Subroutines or BLAS.

Level	Example	# mem refs	# flops	q
1	xAXPY: y = y+αx	3n	$2n^1$	2/3
2	xGEMV: y=y+Ax	n^2	$2n^2$	2
3	xGEMM: C=C+AB	$4n^2$	$2n^3$	n/2

Pattern 2: Spectral Methods

Problem: Systems that are defined in terms of more than one representation.

Solution: Changing the representation of a system can convert a difficult problem into a straightforward algebraic problem

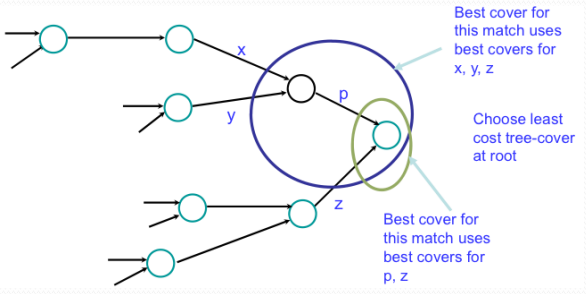


- Fast transform algorithms like the FFT are notoriously difficult to optimize:
  - Luckily, implementations of the FFT exist for every platform:
    - FFTW and SPIRAL: Highly successful auto-tuners for FFT (and others) on PCs and workstations
    - CUFFT for CUDA on NVIDIA GPUs

Pattern 3: Dynamic Programming

Problem: Some search problems have the additional characteristic that the solution to a problem of size *N* can always be assembled out of solutions to problems of size ≤ *N-1*.

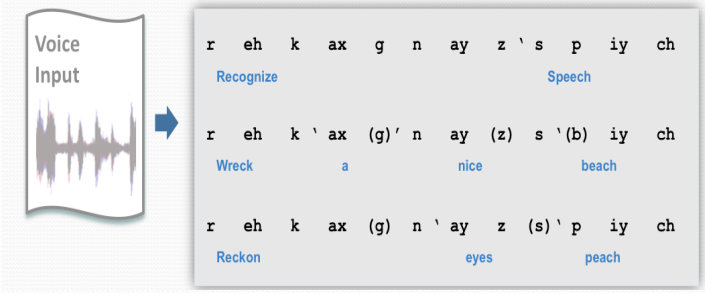
Solution: Explore the search space by finding solutions intoiut; incrementally and not looking for solutions to larger problems until t he solutions to relevant sub- problems are found



Outline

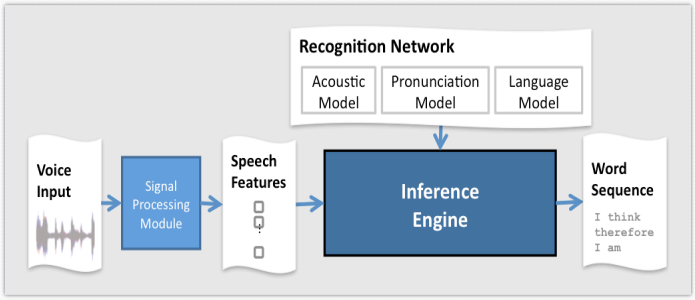
- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern
- Applications to Your Term Projects

Continuous Speech Recognition



- Challenges:
  - Recognizing words from a large vocabulary arranged in exponentially many possible permutations
  - Inferring word boundaries from the context of neighboring words
- Hidden Markov Model (HMM) is the most successful approach

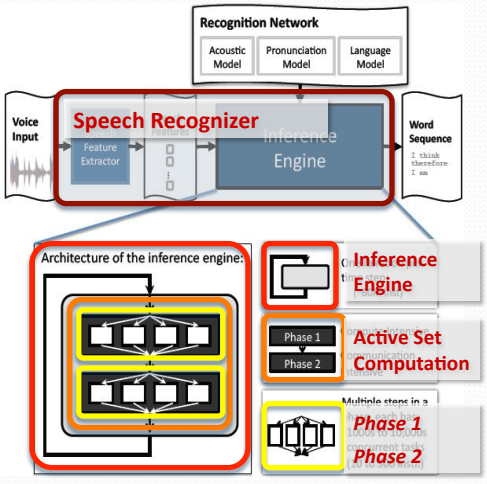
Continuous Speech Recognition



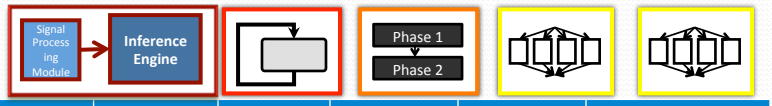
- Inference engine based system
  - Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CSRC, Japan)
- Modular and flexible setup
  - Shown to be effective for Arabic, English, Japanese, and Mandarin

Software Architecture

- A **Hierarchical** Structure:
  - Speech Recognizer
  - Inference Engine
  - Active Set Computation
    - Phase 1: Observation Probability Computation
    - Phase 2: Graph Traversal



GPU-based Implementation



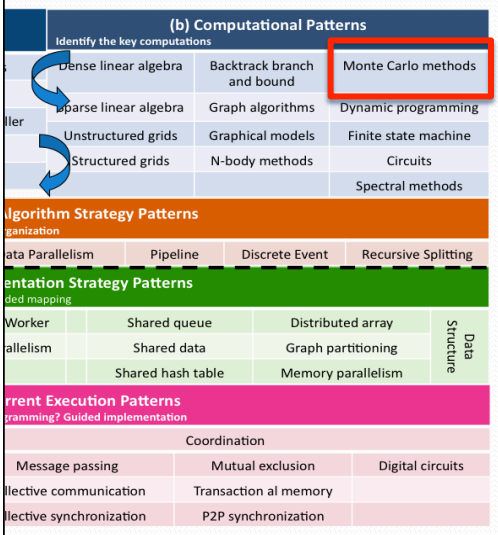
	Speech Recognizer	Inference Engine	Active set Computation	Phase 1: Observation Probability Compute	Phase 2: Graph Traversal
Structural/ Computational Pattern	Pipe and filter Graphical Models	Iterator Dynamic Programming	Pipe and filter	Pipe and filter MapReduce	Graph Algorithm Pipe and filter MapReduce
Algorithm Pattern	Task Parallelism Pipeline	N/A*	N/A*	Task Parallelism	Task Parallelism
Implementation Pattern	Shared Queue	N/A*	N/A*	Strict Data Parallelism	Strict Data Parallelism Shared Queue Shared Data
Execution Pattern	Mutual Exclusion	N/A*	N/A*	SIMD	Mutual Exclusion SIMD

* = data dependency in application resulting in sequential execution

Outline

- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern – Monte Carlo Methods
- Applications to Your Term Projects

OPL: The Structure



Structure of a Software Pattern

Name:
Problem:
Context:
Forces:
- Universal and implementation trade-offs

Solution:
- Structure and consideration

Invariant:
Examples:
- Pedagogical and real world examples

Known Uses:
- Scientific and industrial example

Related Patterns:

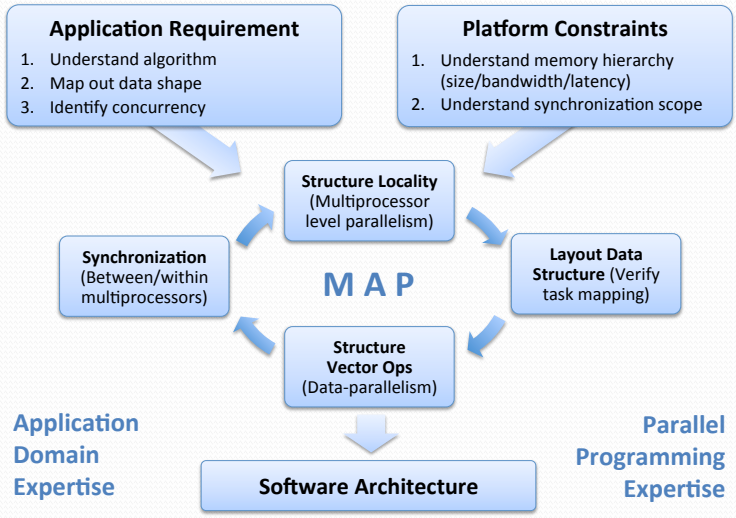
Outline

- Guide to Parallel Software Patterns
- History of Patterns in Programming
- Category of Patterns
  - Structural Patterns
  - Computational Patterns
- Example Use in an Application
- Internal Components of a Pattern
- Applications to Your Term Projects

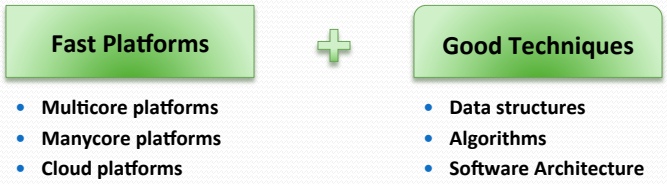
For Your Term Project

- A project to accelerate an application of your choice?
  - **Propose** a small research project
  - **Define** the problem
  - **Design and implement** an **accelerated solution** to this problem
  - **Measure and evaluate** the speedup for the accelerated approaches
  - **Present** the results
  - **Write and hand in** a project proposal, intermediate and final reports

Evaluation of Parallelization Opportunities



How is this relevant to writing fast code?



- Multicore platforms
- Manycore platforms
- Cloud platforms

- Data structures
- Algorithms
- Software Architecture

- Now we have touched on all these aspects of writing fast code
- Can you now **FLIRT** with writing fast code?
  1. **F**eel comfortable hacking up a solution
  2. **L**everage existing software building blocks
  3. **I**ndicate which platform is the best one to use
  4. **R**eason about why a piece of existing code is slow
  5. **T**ake care of potential performance bottlenecks

Can You Answer These Questions Now?

- What are parallel software patterns?
- What are the three goals the software patterns aim to achieve?

- What is a software architecture?
- How is it important for writing fast code?
- What are the five categories of patterns in OPL?

- What are the nine sections in an OPL pattern?

- What are the areas of consideration for your Term Project?