Architecting Parallel Software with Patterns

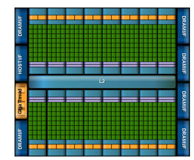
Kurt Keutzer, EECS, Berkeley

with thanks to Tim Mattson, Intel
and the PALLAS team

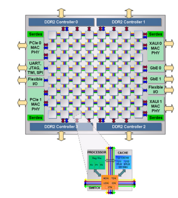
The Challenge of Parallelism

Programming parallel processors is one of the challenges of our era

NVIDIA Tegra 2 system on a chip (SoC)
- Dual-core ARM Cortex A9.
- Integrated GPU. Lots of DSP.
- 1 GHz.
- 2 single-precision GFLOPs peak (CPUs only)



Nvidia Fermi
- 16 cores, 48-way multithreaded,
- 4-wide Superscalar, dual-issue, 3
- 2-wide SIMD (half-pumped)
- 2 MB (16 x 128 KB) Registers, 1
- MB (16 x 64 KB) L1 cache, 0.75 MB L2 Cache

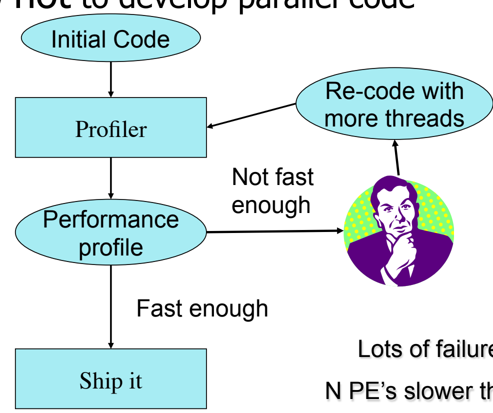


Tilera Tile64
- 64 processors
- Each tile has L1, L2, can run OS
- 443 billion operations/sec.
- 500-833 MHz
- 50 Gbytes/sec memory bandwidth

Outline

- **What doesn't work**
- **Pieces of the problem … and solution**
- **General approach to architecting parallel sw**
- **Detail on Structural Patterns**
- **Detail on Computational Patterns**
- **High-level examples of architecting applications**

Assumption #1:
How not to develop parallel code



Lots of failures

N PE's slower than 1

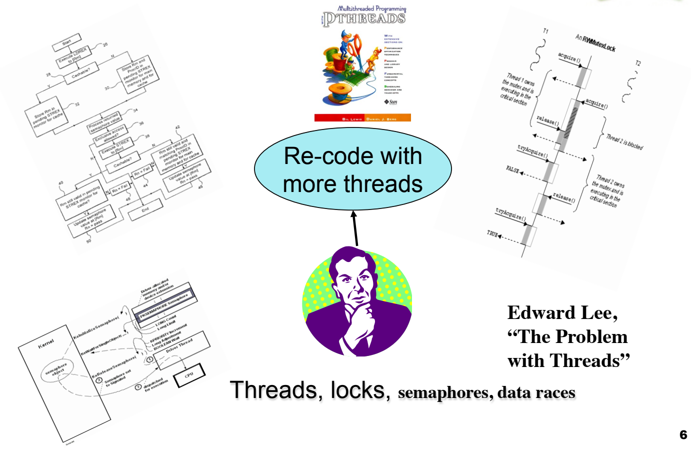
4

Steiner Tree Construction Time By Routing Each Net in Parallel

Benchmark	Serial	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads
adaptec1	1.68	1.68	1.70	1.69	1.69	1.69
newblue1	1.80	1.80	1.81	1.81	1.81	1.82
newblue2	2.60	2.60	2.62	2.62	2.62	2.61
adaptec2	1.87	1.86	1.87	1.88	1.88	1.88
adaptec3	3.32	3.33	3.34	3.34	3.34	3.34
adaptec4	3.20	3.20	3.21	3.21	3.21	3.21
adaptec5	4.91	4.90	4.92	4.92	4.92	4.92
newblue3	2.54	2.55	2.55	2.55	2.55	2.55
average	1.00	1.0011	1.0044	1.0049	1.0046	1.0046

5

Hint: What is this person thinking of?



6

So What's the Alternative?

Outline

- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

8

Principles of SW Design

After 15 years in industry, at one time overseeing the technology of 25 software products, my best principle to facilitate good software design is modularity:

Modularity helps:

- Architect: Makes overall design sound and comprehensible
- Project manager:
  - As a manager I am able to comfortably assign different modules to different developers
  - I am also able to use module definitions to track development
    - Build a PERT chart for development progress
    - Build a “control panel” for current software quality
- Module implementors: As a module implementor I am able to focus on the implementation, optimization, and verification of my module with a minimum of concern about the rest of the design
- Modularity helps to identify key computations

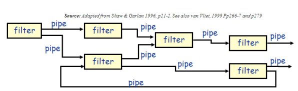
What’s life like without modularity?

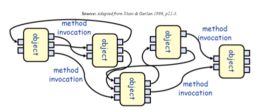
- Spaghetti code
- Wars over the interpretation of the specification
- Waiting on other coders
- Wondering why you didn’t touch anything and now your code broke
- Hard to verify your code in isolation, and therefore hard to optimize
- Hard to parallelize without identifying key computations

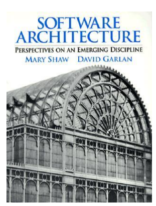
- Modularity will help us obviate all these
  - Parnas, “On the criteria to be used on composing systems into modules,” CACM, December 1972.

Big Step: Architectural Styles (Garland and Shaw, 1996)

- Pipe and filter
- Object oriented
- Event based
- Layered
- Agent and repository
- Process control







Object-Oriented Programming is Not Enough

Focused on:

- Program modularity
- Data locality
- Architectural styles
- Design patterns

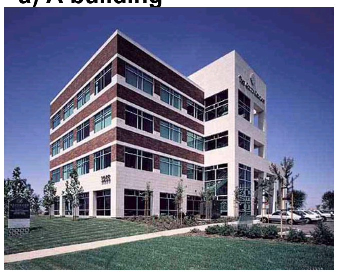
Neglected:

- Application concurrency
- Computational details
- Parallel implementations

12

What's missing?: Is an executing software program more like?

a) A building



b) A factory

We need to consider the machinery – but what is the machinery?

Computations are the Machinery

HPC knows a lot about computations, application concurrency, efficient programming, and parallel implementation

$$x_c \leftarrow \sum_j g_{cj} * x_j$$

$$x \leftarrow W S_\lambda \{W^* x\}$$

$$\boxed{\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.}$$

$$x \leftarrow F(P^T y + P_c^T P_c F^* x)$$

$$\begin{aligned} \text{minimize } & \|Wx\|_1 \\ \text{s.t } & \mathbf{F}_\Omega x = y, \\ & \|\mathbf{G}x - x\|_2 < \varepsilon \end{aligned}$$

$$\begin{aligned} J(w) = & \int_\Omega \psi_1(|I(x+w) - I(x)|^2)dx + \\ & \gamma \int_\Omega \psi_2(|\nabla I(x+w) - \nabla I(x)|^2)dx + \\ & \alpha \int_\Omega \psi_3(|\nabla u|^2 + |\nabla v|^2)dx \end{aligned}$$

14

COMPUTATIONAL RESEARCH DIVISION

Defining Software Requirements for Scientific Computing

Phillip Colella
Applied Numerical Algorithms Group
Lawrence Berkeley National Laboratory

COMPUTATIONAL RESEARCH DIVISION

High-end simulation in the physical sciences consists of seven algorithms:

- Structured Grids (including locally structured grids, e.g. AMR)
- Unstructured Grids
- Fast Fourier Transform
- Dense Linear Algebra
- Sparse Linear Algebra
- Particles
- Monte Carlo

Well-defined targets from algorithmic and software standpoint.

Remainder of this talk will consider one of them (structured grids) in detail.

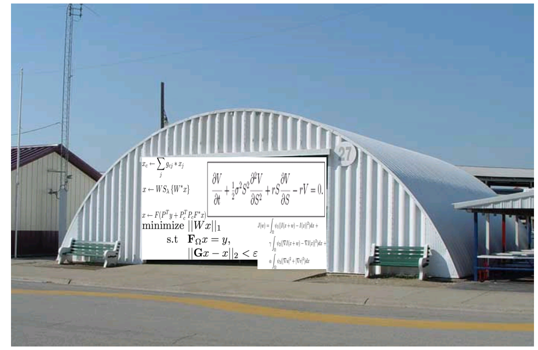
Par Lab's contribution: from 7 to 13 families of computations

Dwarves \ Apps	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												

Understanding computations is not enough either

Unfortunately … HPC approach to software architecture architecture

Technically this is known as a *monolithic* architecture



18

How can we integrate these insights?

- **We wish to find an approach to building software that gives equal support for two key problems of software design – how to structure the software and how to efficiently implement the computations**

19

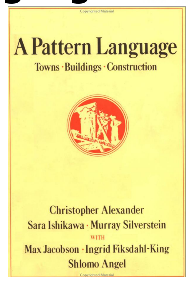
Outline

- **What doesn't work**
- **Pieces of the problem … and solution**
- **General approach to architecting parallel sw**
- **Detail on Structural Patterns**
- **Detail on Computational Patterns**
- **High-level examples of architecting applications**

20

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!!!



21

Uses of Patterns

- Patterns give names and definitions to key elements of design
- This enables us to better:
  - Teach design – a palette of defined design principals
    - Gives ideas to new programmers – approaches you may not have considered
    - Gives a set of finiteness to experienced programmers – if you've considered all the patterns then you can rest assured you've considered the key approaches
  - Guide design – articulate design decisions succinctly
  - Communicate design – improve documentation, facilitate maintenance of software

Uses of Patterns

- Patterns capture and preserve bodies of knowledge about key design decisions
  - Useful implementation techniques
  - Likely challenges/bottlenecks that will come with the use of this pattern (e.g. repository bottleneck in agent and repository)

Architecting Parallel Software with Patterns

Identify the Software Structure

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

Identify the Key Computations

- Graph Algorithms
- Dynamic programming
- Dense/Spare Linear Algebra
- (Un)Structured Grids
- Graphical Models
- Finite State Machines
- Backtrack Branch-and-Bound
- N-Body Methods
- Circuits
- Spectral Methods

24

Architecting Parallel Software

Decompose Tasks
- **Group tasks**
- **Order Tasks**

Decompose Data
- **Identify data sharing**
- **Identify data access**

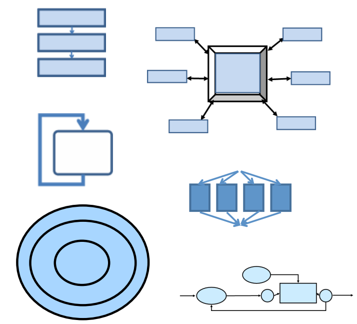
Identify the Software Structure

Identify the Key Computations

25

Identify the SW Structure

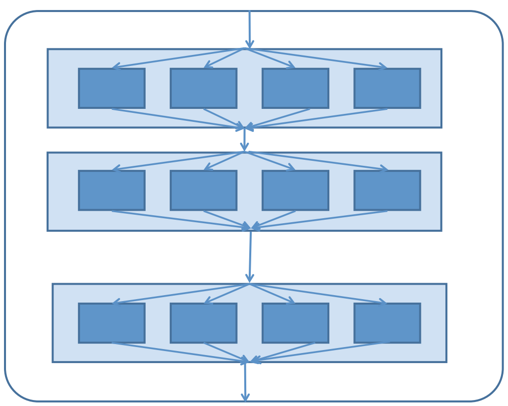
Structural Patterns
- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Process Control
- Layered Systems
- Model-view controller
- Iterator
- MapReduce
- Arbitrary Task Graphs
- Puppeteer



These define the structure of our software but they *do not describe* what is computed

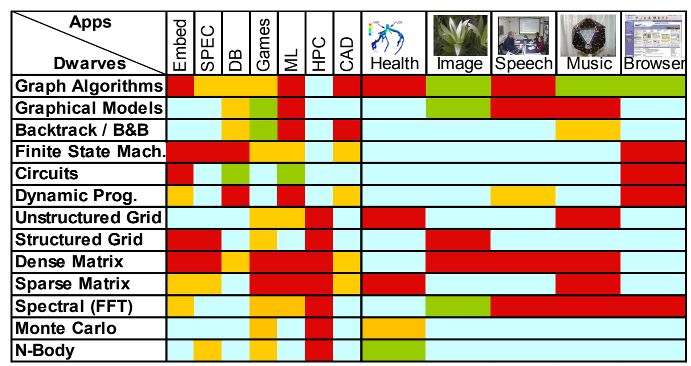
26

Analogy: Layout of Factory Plant

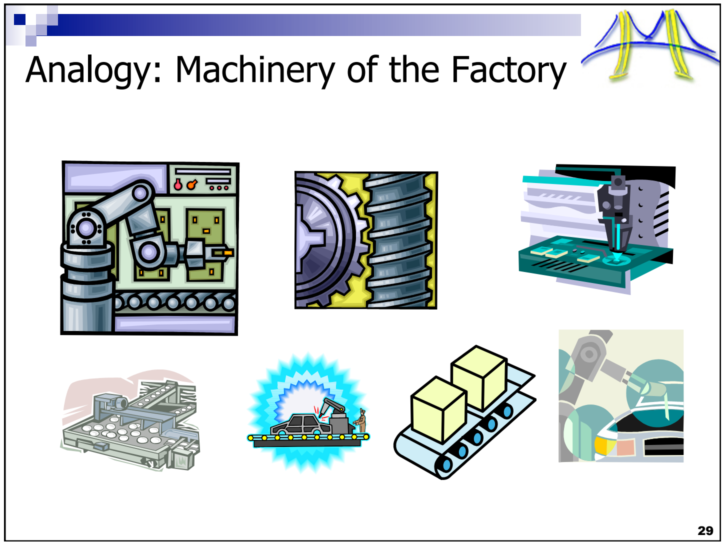


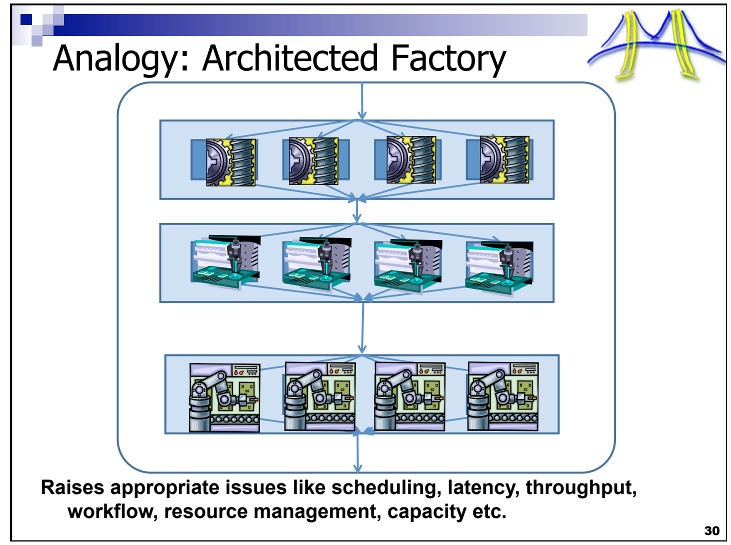
27

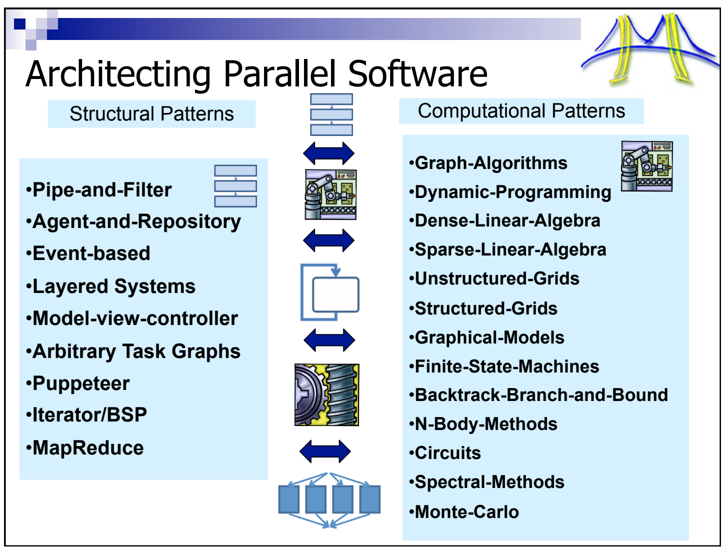
Identify key computations ….

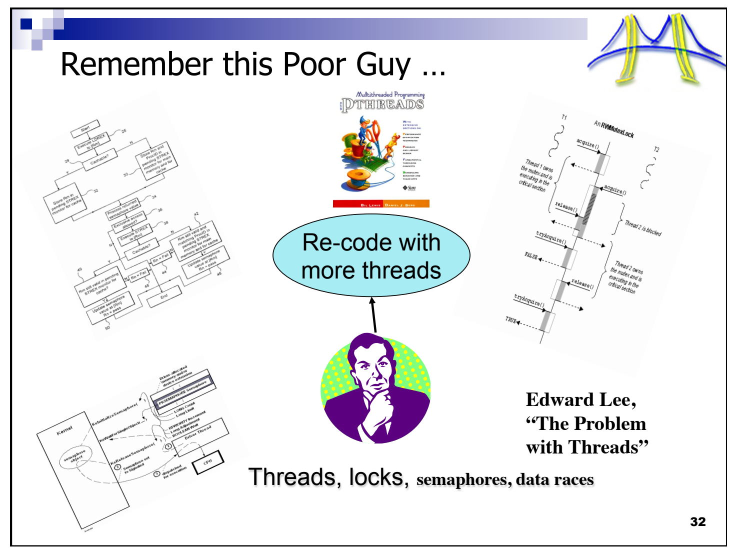


Computational patterns describe the key computations but not how they are implemented


Analogy: Machinery of the Factory
29

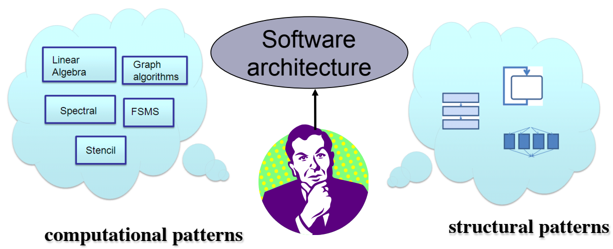

Analogy: Architected Factory
Raises appropriate issues like scheduling, latency, throughput, workflow, resource management, capacity etc.
30


Architecting Parallel Software
Structural Patterns
•Pipe-and-Filter
•Agent-and-Repository
•Event-based
•Layered Systems
•Model-view-controller
•Arbitrary Task Graphs
•Puppeteer
•Iterator/BSP
•MapReduce
Computational Patterns
•Graph-Algorithms
•Dynamic-Programming
•Dense-Linear-Algebra
•Sparse-Linear-Algebra
•Unstructured-Grids
•Structured-Grids
•Graphical-Models
•Finite-State-Machines
•Backtrack-Branch-and-Bound
•N-Body-Methods
•Circuits
•Spectral-Methods
•Monte-Carlo


Remember this Poor Guy …
Re-code with more threads
Edward Lee, "The Problem with Threads"
Threads, locks, semaphores, data races
32

What's this person thinking of …?

- Need to integrate the insights into computation provided by HPC with the insights into program structure provided by software architectural styles



33

Outline

- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications

34

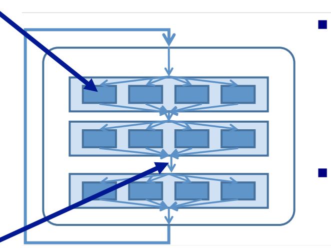
Inventory of Structural Patterns

1. pipe and filter
2. iterator
3. MapReduce
4. blackboard/agent and repository
5. process control
6. Model View Controller
7. layered
8. event-based coordination
9. puppeteer
10. static task graph

35

Elements of a structural pattern

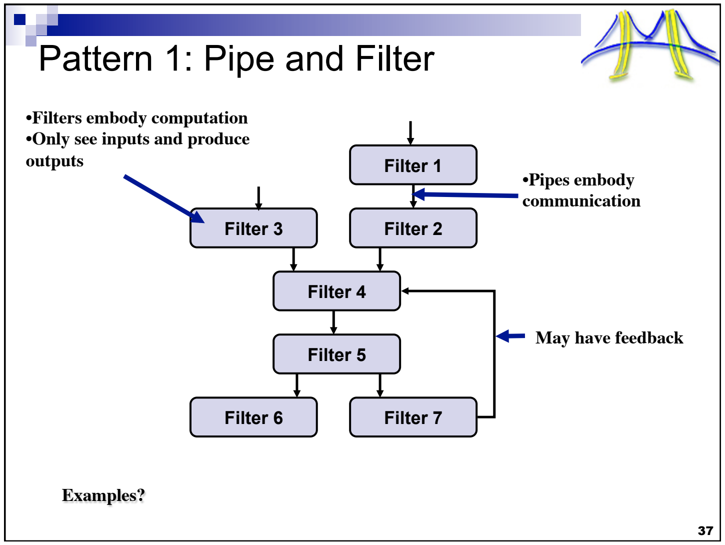
- Components are where the computation happens

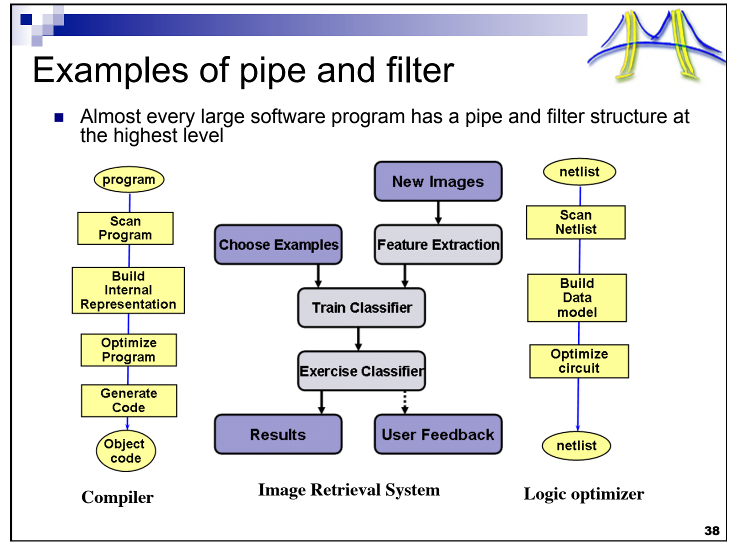


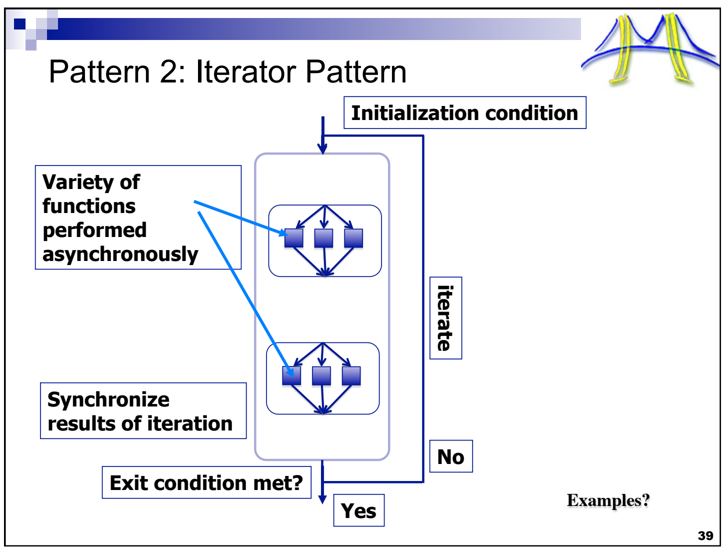
- A configuration is a graph of components (vertices) and connectors (edges)
- A structural patterns may be described as a familiy of graphs.

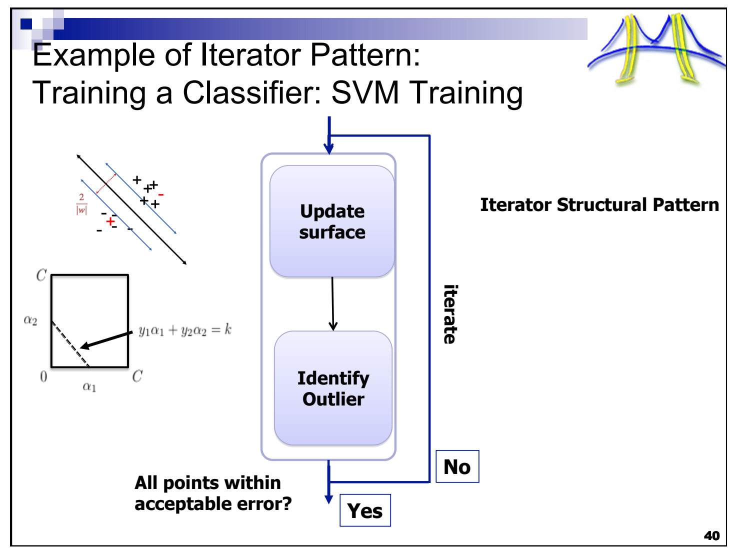
Connectors are where the communication happens

36


Pattern 1: Pipe and Filter
•Filters embody computation
•Only see inputs and produce outputs
Filter 1
Filter 3
Filter 2
Filter 4
Filter 5
Filter 6
Filter 7
•Pipes embody communication
May have feedback
Examples?
37


Examples of pipe and filter
Almost every large software program has a pipe and filter structure at the highest level
program
Scan Program
Build Internal Representation
Optimize Program
Generate Code
Object code
Compiler
New Images
Choose Examples
Feature Extraction
Train Classifier
Exercise Classifier
Results
User Feedback
Image Retrieval System
netlist
Scan Netlist
Build Data model
Optimize circuit
netlist
Logic optimizer
38

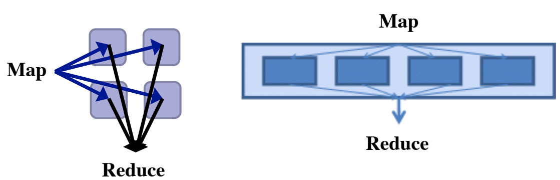

Pattern 2: Iterator Pattern
Initialization condition
Variety of functions performed asynchronously
iterate
Synchronize results of iteration
No
Exit condition met?
Yes
Examples?
39


Example of Iterator Pattern:
Training a Classifier: SVM Training
Update surface
Identify Outlier
iterate
Iterator Structural Pattern
No
All points within acceptable error?
Yes
40

Pattern 3: MapReduce

To us, it means

- A map stage, where data is mapped onto independent computations
- A reduce stage, where the results of the map stage are summarized (i.e. reduced)

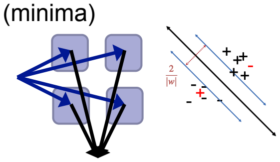


Examples?

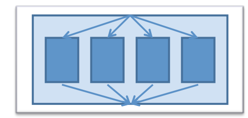
41

Examples of Map Reduce

- General structure:
- Map a computation across distributed data sets
- Reduce the results to find the best/(worst), maxima/(minima)



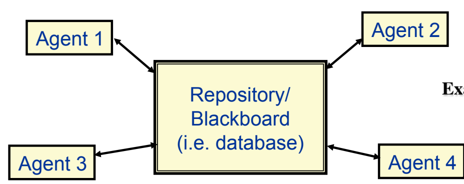
Support-vector machines (ML)
• Map to evaluate distance from the frontier
• Reduce to find the greatest outlier from the frontier



Speech recognition
• Map HMM computation to evaluate word match
• Reduce to find the most-likely word sequences

42

Pattern 4: Agent and Repository



Examples?

Agent and repository : Blackboard structural pattern

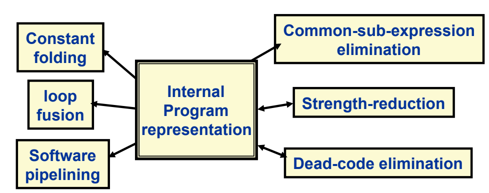
Agents cooperate on a shared medium to produce a result

Key elements:

- **Blackboard**: repository of the resulting creation that is shared by all agents (circuit database)
- **Agents**: intelligent agents that will act on blackboard (optimizations)
- **Manager**: orchestrates agents access to the blackboard and creation of the aggregate results (scheduler)

43

Example: Compiler Optimization

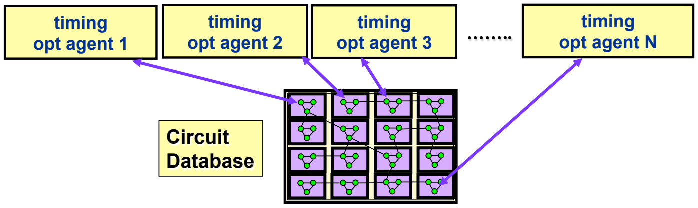


Optimization of a software program

- Intermediate representation of program is stored in the repository
- Individual agents have heuristics to optimize the program
- Manager orchestrates the access of the optimization agents to the program in the repository
- Resulting program is left in the repository

44

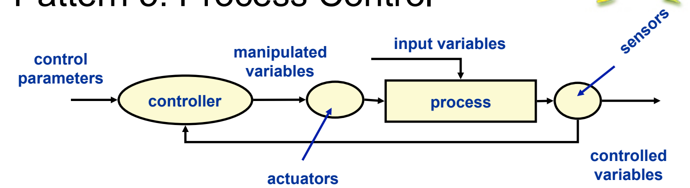
Example: Logic Optimization



- Optimization of integrated circuits
- Integrated circuit is stored in the repository
- Individual agents have heuristics to optimize the circuitry of an integrated circuit
- Manager orchestrates the access of the optimization agents to the circuit repository
- Resulting optimized circuit is left in the repository

45

Pattern 5: Process Control



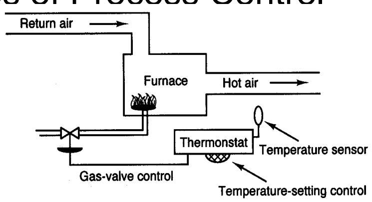
Source: Adapted from Shaw & Garlan 1996, p27-31.

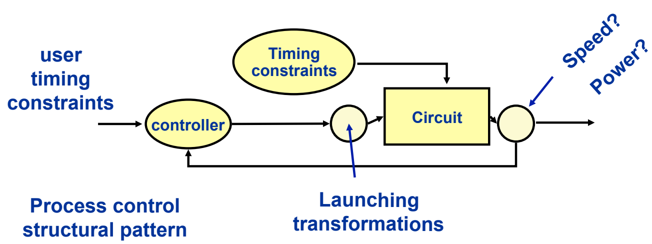
- Process control:
  - **Process**: **underlying phenomena to be controlled/computed**
  - **Actuator**: **task(s) affecting the process**
  - **Sensor**: **task(s) which analyze the state of the process**
  - **Controller**: **task which determines what actuators should be effected**

Examples?

46

Examples of Process Control

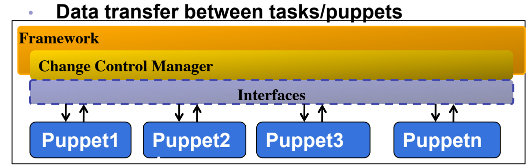




47

Pattern 9: Puppeteer

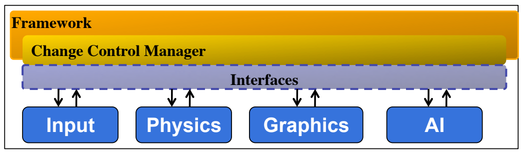
- **Need an efficient way to manage and control the interaction of multiple simulators/computational agents**
- **Puppeteer Pattern – guides the interaction between the tasks/ puppets to guarantee correctness of the overall task**
- **Puppeteer: 1) schedules puppets 2) manages exchange of data between puppets**
- **Difference with agent and repository?**
  - **No central repository**
  - **Data transfer between tasks/puppets**



Examples?

48/17

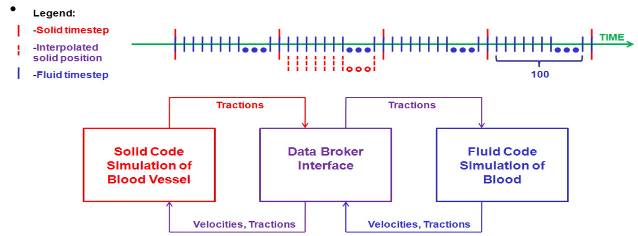
Video Game



49/17

Model of circulation

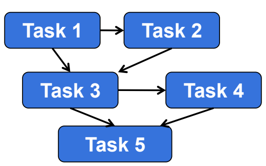
- Modeling of blood moving in blood vessels
- The computation is structured as a controlled interaction between solid (blood vessel) and fluid (blood) simulation codes
- The two simulations use different data structures and the number of iterations for each simulation code varies
- Need an efficient way to manage and control the interaction of the two codes



50

Pattern 10: Static Task Graph

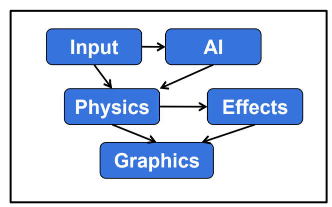
- Tasks receive inputs and produce outputs
- All data sharing is through explicit messaging (arrow "→" means message passing communication)
- Task configuration is statically defined and may not be changed at runtime



Example?

Example: one game architecture

- There exist fixed dependencies between subsystems
- Can be modeled as an arbitrary task graph
- Example: Moving the zombie
  - Keyboard -> AI -> Physics -> Graphics

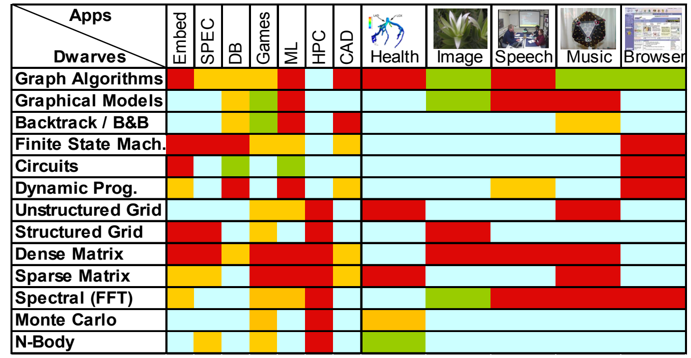


Outline

- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- → Detail on Computational Patterns
- High-level examples of architecting applications

53

You explore these every class

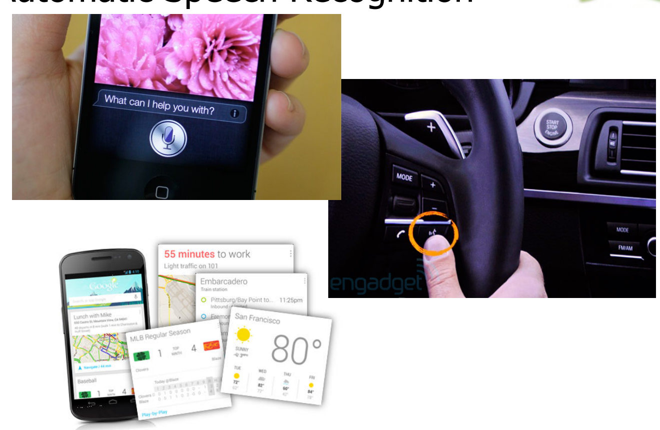


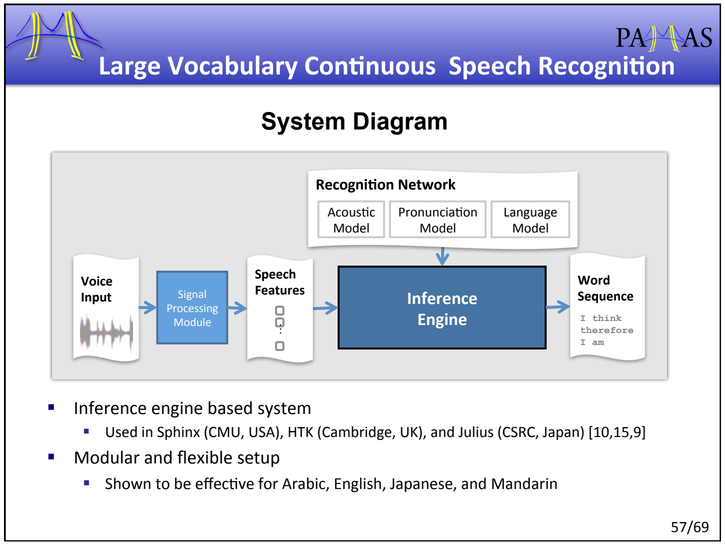
Outline

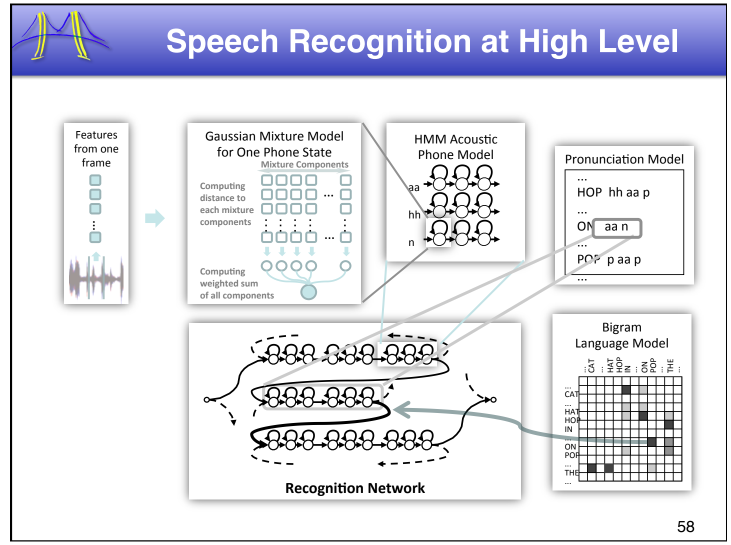
- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- → High-level examples of architecting applications

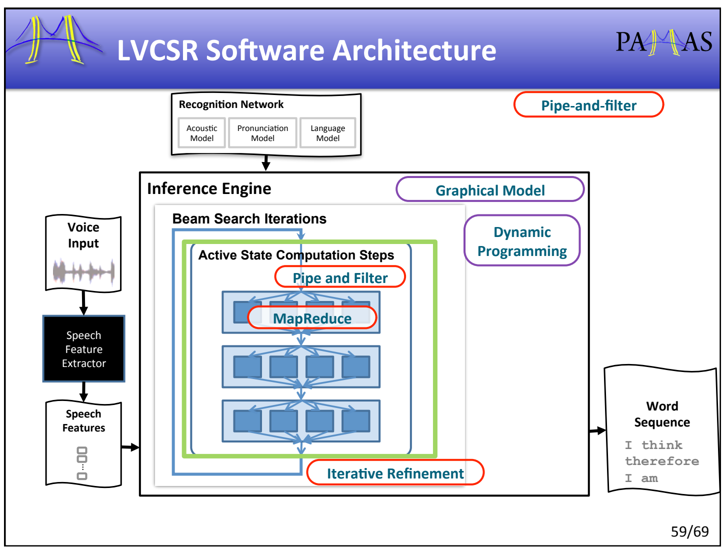
55

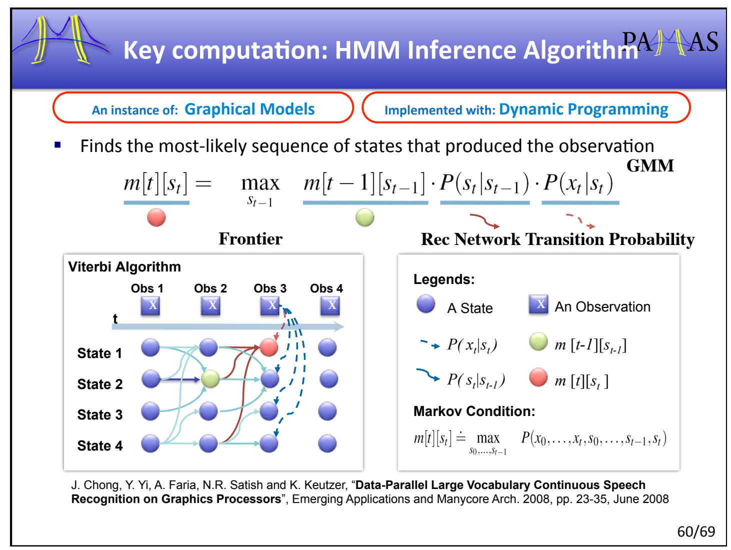
Automatic Speech Recognition



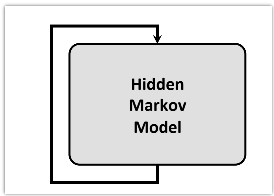

PALLAS
Large Vocabulary Continuous Speech Recognition
System Diagram
Recognition Network
Acoustic Model
Pronunciation Model
Language Model
Voice Input
Signal Processing Module
Speech Features
Inference Engine
Word Sequence
I think therefore I am
Inference engine based system
Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CSRC, Japan) [10,15,9]
Modular and flexible setup
Shown to be effective for Arabic, English, Japanese, and Mandarin
57/69


Speech Recognition at High Level
Features from one frame
Gaussian Mixture Model for One Phone State
Mixture Components
Computing distance to each mixture components
Computing weighted sum of all components
HMM Acoustic Phone Model
aa
hh
n
Pronunciation Model
HOP hh aa p
ON aa n
POP p aa p
Bigram Language Model
Recognition Network
58


LVCSR Software Architecture
PALLAS
Recognition Network
Acoustic Model
Pronunciation Model
Language Model
Pipe-and-filter
Inference Engine
Graphical Model
Beam Search Iterations
Dynamic Programming
Active State Computation Steps
Pipe and Filter
MapReduce
Iterative Refinement
Voice Input
Speech Feature Extractor
Speech Features
Word Sequence
I think therefore I am
59/69


Key computation: HMM Inference Algorithm
PALLAS
An instance of: Graphical Models
Implemented with: Dynamic Programming
Finds the most-likely sequence of states that produced the observation
$m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$
GMM
Frontier
Rec Network Transition Probability
Viterbi Algorithm
Obs 1
Obs 2
Obs 3
Obs 4
t
State 1
State 2
State 3
State 4
Legends:
A State
An Observation
$P(x_t|s_t)$
$m[t-1][s_{t-1}]$
$P(s_t|s_{t-1})$
$m[t][s_t]$
Markov Condition:
$m[t][s_t] \doteq \max_{s_0,\ldots,s_{t-1}} P(x_0,\ldots,x_t,s_0,\ldots,s_{t-1},s_t)$
J. Chong, Y. Yi, A. Faria, N.R. Satish and K. Keutzer, "Data-Parallel Large Vocabulary Continuous Speech Recognition on Graphics Processors", Emerging Applications and Manycore Arch. 2008, pp. 23-35, June 2008
60/69

Iterative Refinement Structural Pattern



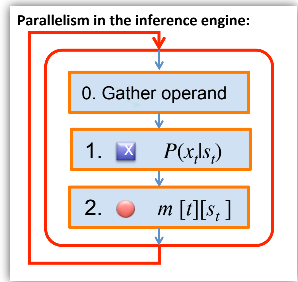
- One iteration per time step
- Identify the set of probable states in the network given acoustic signal given current active state set
- Prune unlikely states
- Repeat

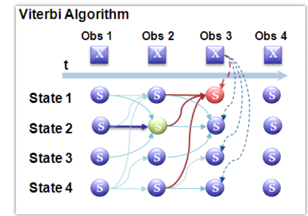
Structural Patterns	Model-View-Controller
Pipe-and-Filter	Iterative-Refinement
Agent-and-Repository	Map-Reduce
Process-Control	Layered-Systems
Event-Based/Implicit-Invocation	Arbitrary-Static-Task-Graph
Puppeteer	

61/69

Inference Engine in LVCSR

- Three steps of inference
  0. Gather operands from irregular data structure to runtime buffer
  1. Perform observation probability computation
  2. Perform graph traversal computation



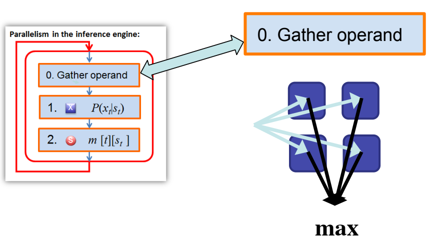


62/69

Each Filter is a Map Reduce
0. Gather operands

$$m[t][s_t] = \max_{s_{t-1}} \; m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$

- Gather and coalesce each of the above operands for every s_t
- Facilitates opportunity for SIMD

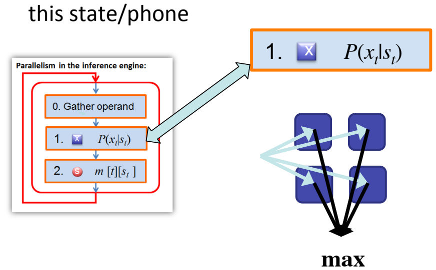


63/69

Each Filter is Map Reduce
1. observation probability computation

$$m[t][s_t] = \max_{s_{t-1}} \; m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot \underline{P(x_t|s_t)}$$

- Gaussian Mixture Model Probability
- Probability that given this feature-frame (e.g. 10ms) we are in this state/phone



64/69

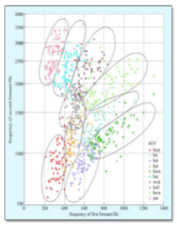
1. Observation Probability Computational Patterns

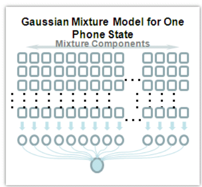
PALLAS

- Observation probabilities are computed from Gaussian Mixture Models
  - Each Gaussian probability in each mixture is independent
  - Probability for one phone state is the sum of all Gaussians times the mixture probability for that state

$$p(x_j \mid \mu_i, \Sigma_i) = \sum_i \pi_i g(x_j \mid \mu_i, \Sigma_i)$$

$$= \sum_i \pi_i \frac{1}{(2\pi)^{\frac{D}{2}} \mid \Sigma_i \mid^{\frac{1}{2}}} exp\{-\frac{1}{2}(x_j - \mu_i)^T \Sigma_i^{-1} (x_j - \mu_i)\}$$





Computational Patterns: Graph-Algorithms, Dynamic-Programming, Dense-Linear-Algebra, Sparse-Linear-Algebra, Unstructured-Grids, Structured-Grids; Graphical-Models, Finite-State-Machines, Backtrack-Branch-and-Bound, N-Body-Methods, Circuits, Spectral-Methods, Monte-Carlo

Dan Klein's CS288, Lecture 9

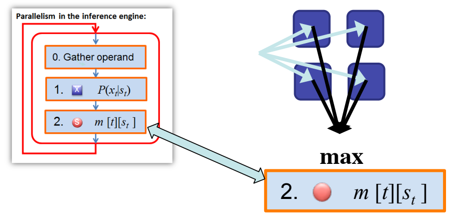
65/69

Each Filter is Map Reduce
2. graph traversal computation

PALLAS

- **Map** probability computation across distributed data sets – perform multiplication as below
- **Reduce** the results to find the maximally likely states

$$m[t][s_t] = \max_{s_{t-1}} \; m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$



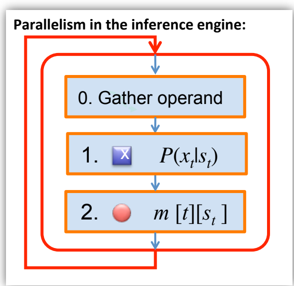
66/69

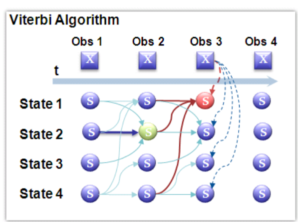
All together: Inference Engine in LVCSR

PALLAS

- Put all together the inference engine is dynamic programming

Computational Patterns: Graph-Algorithms, Dynamic-Programming, Dense-Linear-Algebra, Sparse-Linear-Algebra, Unstructured-Grids, Structured-Grids; Graphical-Models, Finite-State-Machines, Backtrack-Branch-and-Bound, N-Body-Methods, Circuits, Spectral-Methods, Monte-Carlo

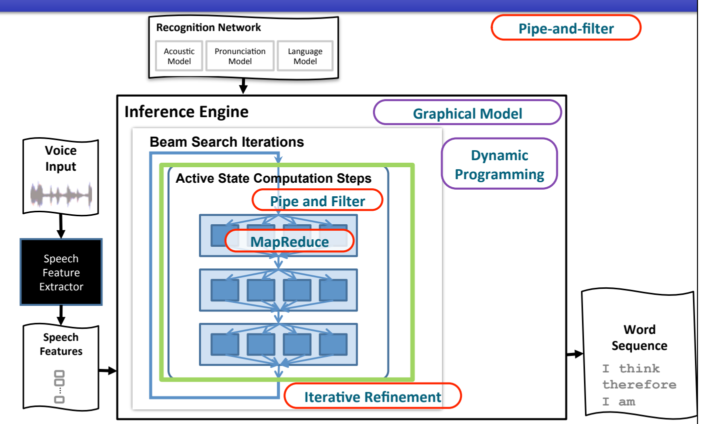




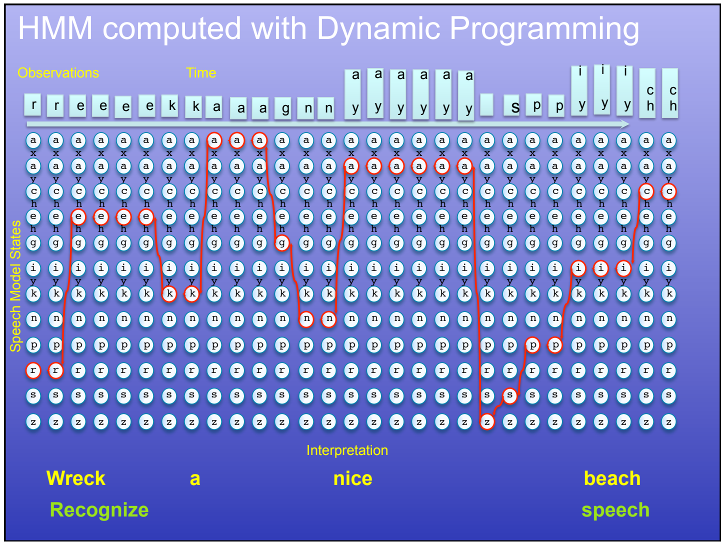
67/69

LVCSR Software Architecture

PALLAS



68/69



This Approach Works

PALLAS

PALLAS has a 10-year history of accelerating critical applications

Application	Our Speedup	
MRI Reconstruction	100x	IEEE TMI 2012
SVM Training	20x	ICML 2008
SVM Deployment	109x	
Value-at-risk	60x	Wiley 2011
Option Pricing	25x	
Speech Recognition	11x	Interspeech 2010, 2011
Video Event Detection	78x	ICMR 2015
Contour Detection	130x	ICCV 2009
Object Recognition	80x	WACV 2011
Poselets	20x	
Optical Flow	32x	ECCV 2010
Deformable Parts Models	45x	CVPR 2015
Image Convolution	8x	ICIP 2013
Deep Neural Network Training	47 - 140x	CVPR 2016

"Considerations When Evaluating Microprocessor Platforms" In Proceedings of the 3rd USENIX conference on Hot topics in parallelism (HotPar'11). USENIX Association, Berkeley, CA, USA.

70/69

Outline

PALLAS

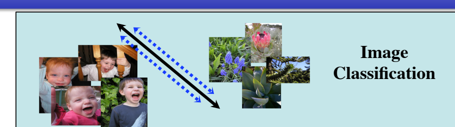
- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications
- ⇨ Summary

71/69

Recap: Architecting Parallel Software

PALLAS

1. Start with a compelling, performance sensitive application.



Catanzaro, Sundaram, Keutzer, "Fast SVM Training and Classification on Graphics Processors", ICML 2008

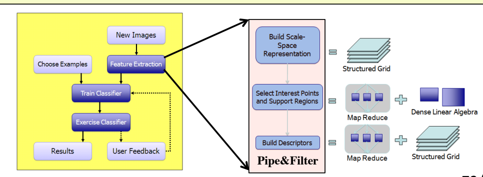
2. Define the overall structure

Identify the Software Structure

Identify the Key Computations

3. Define computations inside structural elements

4. Compose Structural and computational patterns to yield software architecture

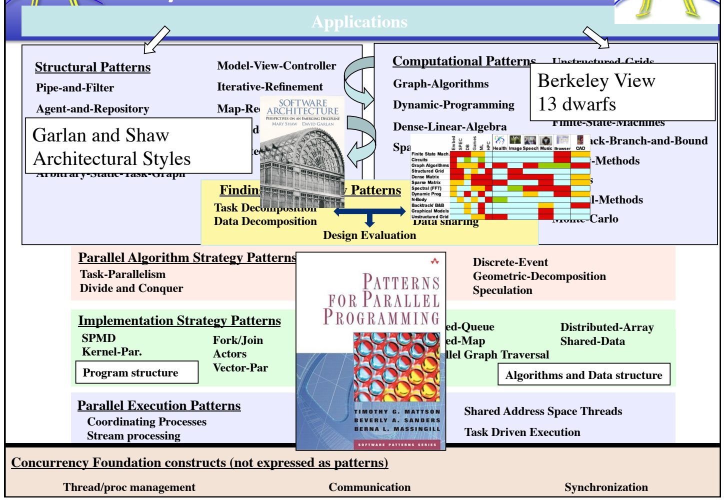


"Image Feature Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC '09

72/69

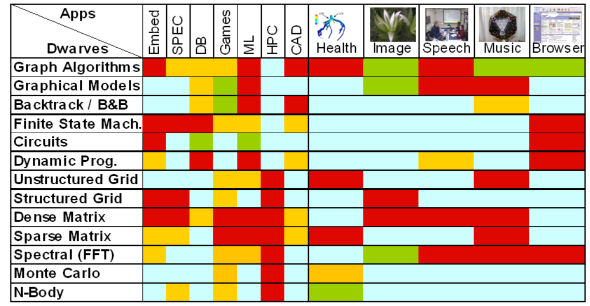
Architecting Paralle Software
Keutzer and Mattson

OPL/PLPP 2012



Computational Patterns Make me Feel Smart

PAMAS



- For many years computation has been like a big ball of yarn
- Computational patterns help us to unravel it into 13 strands
- Alan Kay "Perspective is worth 100 IQ points."
- Computational patterns give us perspective on computation

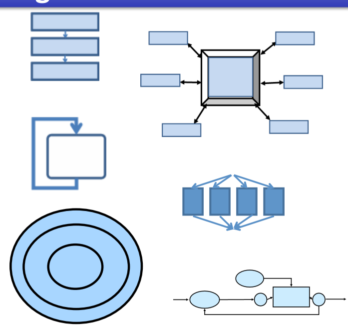
74/69

Structural Patterns Make me Feel Organized

PAMAS

Structural Patterns

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



•The modularity provided by structural patterns make me feel organized.

• **Even the most complex application can be broken down into manageable modules**

75/69

Summary

PAMAS

- The key to productive and efficient parallel programming is creating a good software architecture – a hierarchical composition of:
- Structural patterns: enforce modularity and expose invariants
  - I showed you six – four more will be all you ever need
- Computational patterns: identify key computations to be parallelized
- Orchestration of computational and structural patterns creates architectures which greatly facilitates the development of parallel programs:

Short Course Hosted at Intel Software University Website:
- **Engineering Parallel Software with Patterns**
  - **http://university.intel.com/**

Semester Long Course Taught at Berkeley and Hosted at XSEDE
- **https://cvw.cac.cornell.edu/eps/default**

Patterns: https://patterns.eecs.berkeley.edu/

76/69

More examples

77