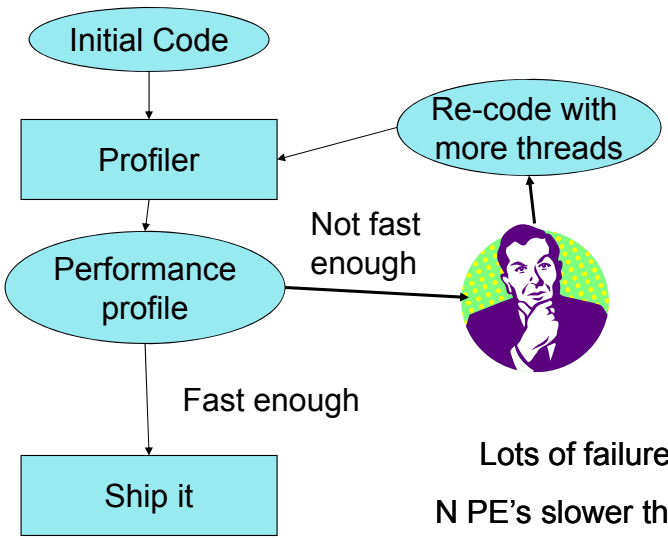
# Architecting Parallel Software with Patterns

Kurt Keutzer,
and
the PALLAS Group
Michael Anderson, Katya Gonina,
Patrick Li, David Sheffield,
Jike Chong (Simply Hired), Mark Murphy (Google)
Bor-yiing Su (Google), Naryanan Sundaram (Intel-Dubey)
Yubei Chen, Forrest Iandola, and Peter Jin
With special thanks to Tim Mattson

## Assumption #1:
## How not to develop parallel code




Lots of failures

N PE's slower than 1

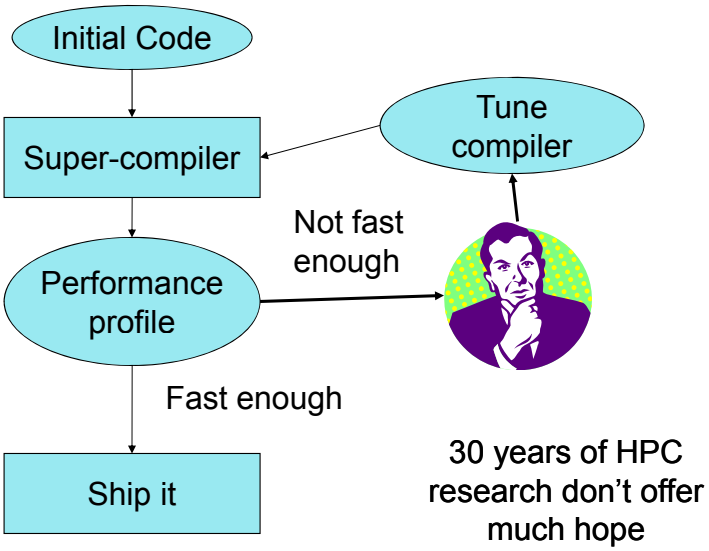
2/66

## 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 |

3/66

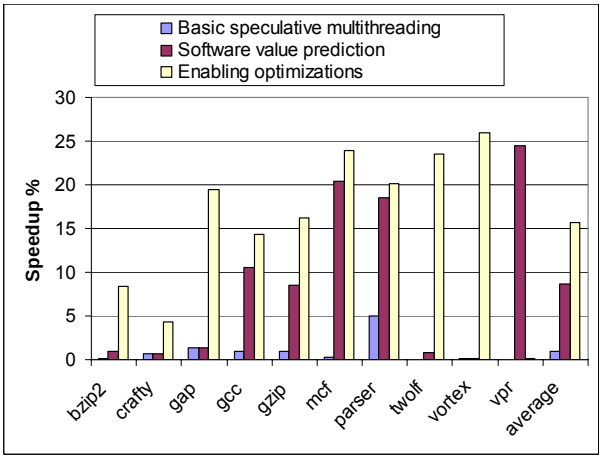
## Assumption #2: Nor this




30 years of HPC research don't offer much hope

4/66

## Automatic parallelization?

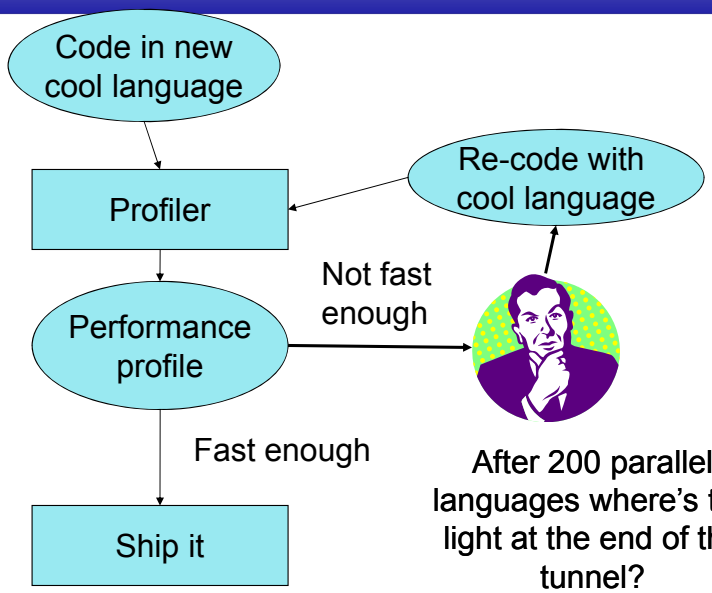



- Aggressive techniques such as speculative multithreading help, but they are not enough.
- Ave SPECint speedup of 8% … will climb to ave. of 15% once their system is fully enabled.
- There are no indications auto par. will radically improve any time soon.
- Hence, I do not believe Auto-par will solve our problems.

Results for a simulated dual core platform configured as a main core and a core for speculative execution.

5/66

## Assumption #3: This won't help either




After 200 parallel languages where's the light at the end of the tunnel?

6/66

## Parallel Programming environments in the 90's

| | | | | | |
|---|---|---|---|---|---|
| ABCPL | CORRELATE | GLU | Mentat | Parafrase2 | pC++ |
| ACE | CPS | GUARD | Legion | Paralation | SCHEDULE |
| ACT++ | CRL | HAsL. | Meta Chaos | Parallel-C++ | SciTL |
| Active messages | CSP | Haskell | Midway | Parallaxis | POET |
| Adl | Cthreads | HPC++ | Millipede | ParC | SDDA. |
| Adsmith | CUMULVS | JAVAR. | CparPar | ParLib++ | SHMEM |
| ADDAP | DAGGER | HORUS | Mirage | ParLin | SIMPLE |
| AFAPI | DAPPLE | HPC | MpC | Parmacs | Sina |
| ALWAN | Data Parallel C | IMPACT | MOSIX | Parti | SISAL. |
| AM | DC++ | ISIS. | Modula-P | pC | distributed smalltalk |
| AMDC | DCE++ | JAVAR | Modula-2* | pC++ | SMI. |
| AppLeS | DDD | JADE | Multipol | PCN | SONiC |
| Amoeba | DICE. | Java RMI | MPI | PCP: | Split-C. |
| ARTS | DIPC | javaPG | MPC++ | PH | SR |
| Athapascan-0b | DOLIB | JavaSpace | Munin | PEACE | Sthreads |
| Aurora | DOME | JIDL | Nano-Threads | PCU | Strand. |
| Automap | DOSMOS. | Joyce | NESL | PET | SUIF. |
| bb_threads | DRL | Khoros | NetClasses++ | PETSc | Synergy |
| Blaze | DSM-Threads | Karma | Nexus | PENNY | Telegrphos |
| BSP | Ease. | KOAN/Fortran-S | Nimrod | Phosphorus | SuperPascal |
| BlockComm | ECO | LAM | NOW | POET. | TCGMSG. |
| C*. | Eiffel | Lilac | Objective Linda | Polaris | Threads.h++. |
| "C* in C | Eilean | Linda | Occam | POOMA | TreadMarks |
| C** | Emerald | JADA | Omega | POOL-T | TRAPPER |
| CarlOS | EPI | WWWinda | OpenMP | PRESTO | uC++ |
| Cashmere | Excalibur | ISETL-Linda | Orca | P-RIO | UNITY |
| C4 | Express | ParLin | OOF90 | Prospero | UC |
| CC++ | Falcon | Eilean | P++ | Proteus | V |
| Chu | Filaments | P4-Linda | P3L | QPC++ | ViC* |
| Charlotte | FM | Glenda | p4-Linda | PVM | Visifold V-NUS |
| Charm | FLASH | POSYBL | Pablo | PSI | VPE |
| Charm++ | The FORCE | Objective-Linda | PADE | PSDM | Win32 threads |
| Cid | Fork | LiPS | PADRE | Quake | WinPar |
| Cilk | Fortran-M | Locust | Panda | Quark | WWWinda |
| CM-Fortran | FX | Lparx | Papers | Quick Threads | XENOOPS |
| Converse | GA | Lucid | AFAPI. | Sage++ | XPC |
| Code | GAMMA | Maisie | Para++ | SCANDAL | Zounds |
| COOL | Glenda | Manifold | Paradigm | SAM | ZPL |

7/66

## So What's the Alternative?

8/66

## Principles of SW Design

- After 15 years in industry, at one time overseeing the techology of 25 software products, my two best principles to facilitate good software design are:
  - Use of modularity
  - Definition of invariants
- 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
  - 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
  - Identify invariants and key computations

9/66

## Non-Principles of SW Design

- 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.
  - But modularity alone is not enough, because …

10/66

## Modularity is important …. But …

PARAS

Pop quiz: Is software more like?

**a) A building** **b) A factory**

11/66

## What computations we do is as important than how we do them ….

PARAS

| 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 | | | | | | | | | | | | |

12/66

## 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, the machinery
  - Structural patterns – the molecular bonds, the layout
- This software architecture naturally gives:
  - Modularity supplied through structural patterns
    - Efficient management
    - Efficient implementation
    - Efficient verification
  - Computational efficiency supplied through computational patterns

13/66

## Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
- Summary

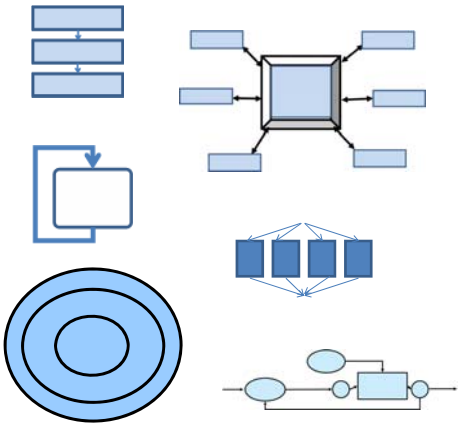
14/66

## Identify the SW Structure

PARMAS

Structural Patterns

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



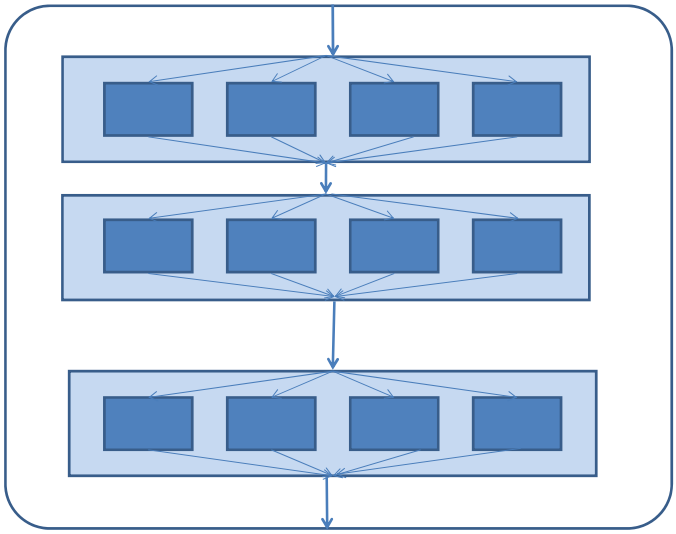
These define the structure of our software but they *do not describe* what is computed

15

15/66

## Analogy: Layout of Factory Plant

PARMAS



16

16/66

## Identify Key Computations

PARLAS

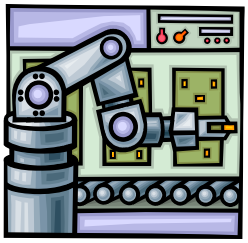
| Apps / Dwarves | Embed | SPEC | DB | Games | ML | HPC | CAD | Health | Image | Speech | Music | Browser |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Graph Algorithms | | | | | | | | | | | | |
| Graphical Models | | | | | | | | | | | | |
| Backtrack / B&B | | | | | | | | | | | | |
| Finite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

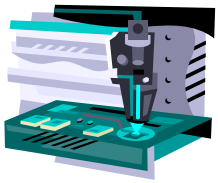
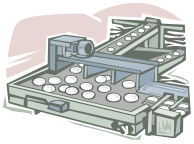
- Computational patterns describe the key computations but not how they are implemented

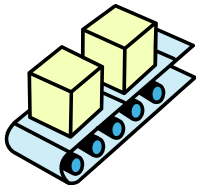
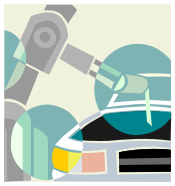
17/66

## Analogy: Machinery of the Factory

PARLAS



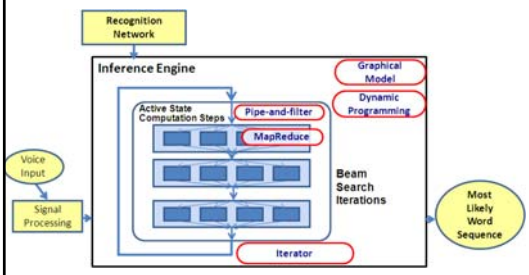



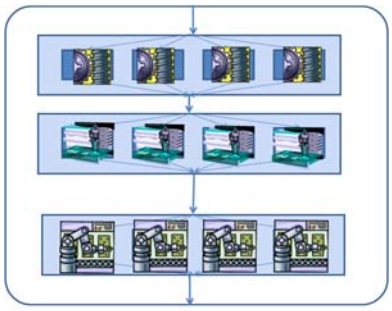



18

18/66

## Architecting the Whole Application



- SW Architecture of Large-Vocabulary Continuous Speech Recognition

Analogous to the design of an entire manufacturing plant

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

19

19/66

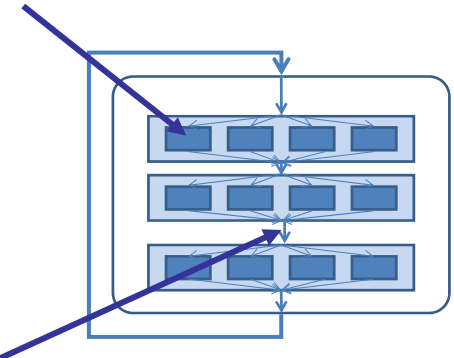
## Outline

- Architecting Parallel Software
- Structural Patterns
  - Pipe and filter
  - Iterator
  - Map Reduce
- Computational Patterns
- Examples
- Summary

20/66

## Elements of a structural pattern

- Components are where the computation happens



Connectors are where the communication happens

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

21/66

## Inventory of Structural Patterns

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

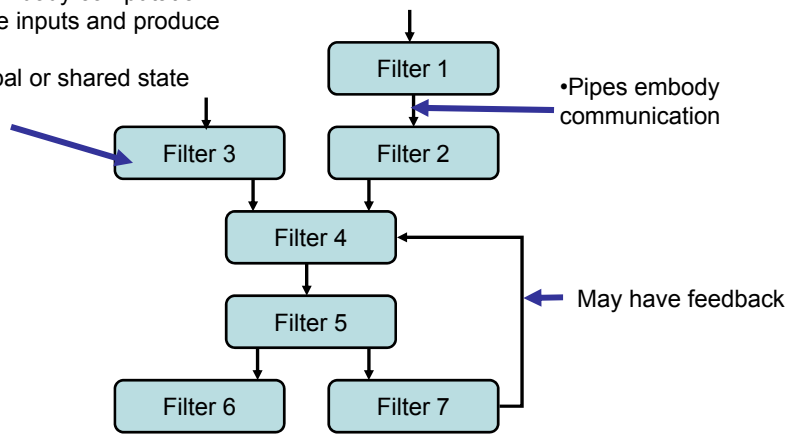
- We build arbitrarily complex software structures out of these nine patterns

22/66

## Pattern 1: Pipe and Filter

PAMAS

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



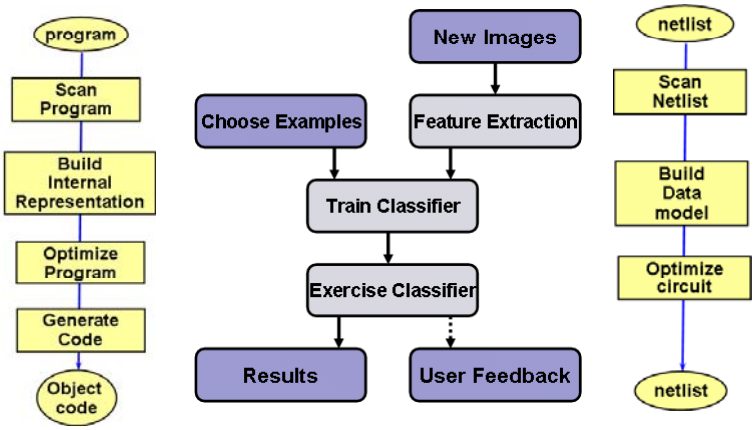

Examples?

23/66

## Examples of pipe and filter

PAMAS

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




Compiler

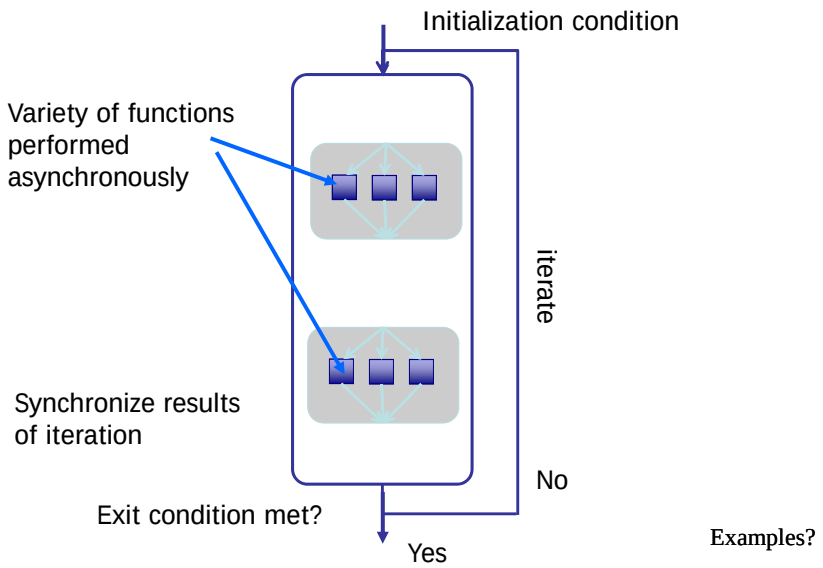
Image Retrieval System

Logic optimizer

24

24/66

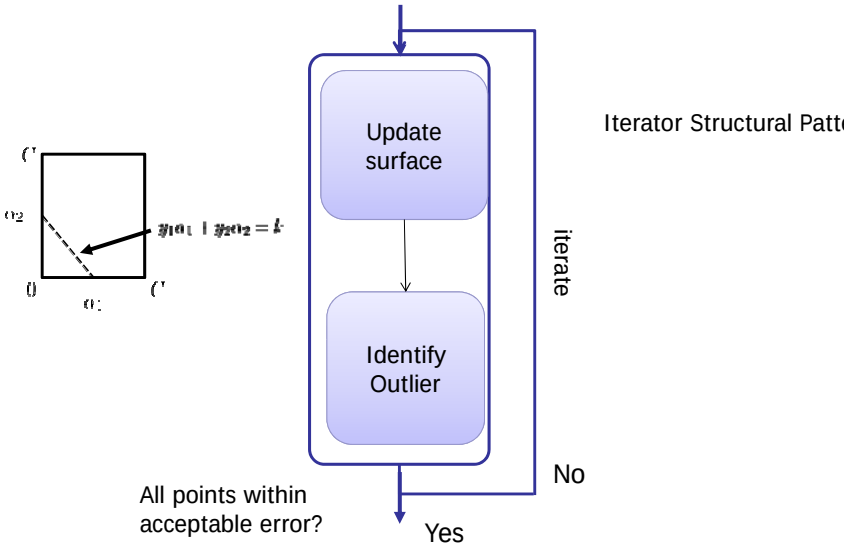
## Pattern 2: Iterator Pattern




Examples?

25/66

## Example of Iterator Pattern: Training a Classifier: SVM Training

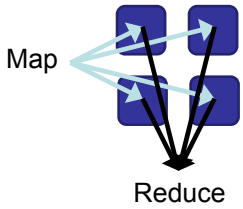


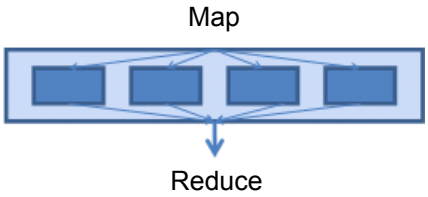

Iterator Structural Pattern

26/66

## 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)



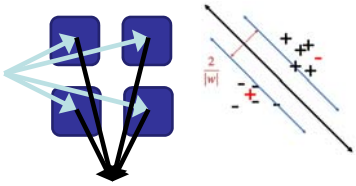




Examples?

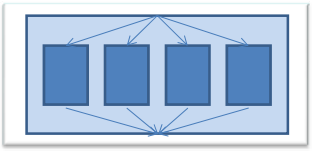
27/66

## 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

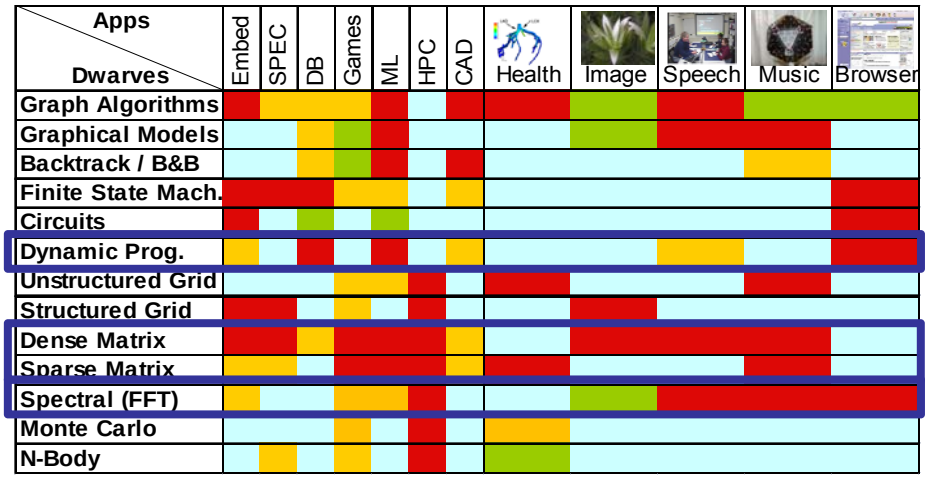
28/66

## Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
  - Linear Algebra
  - Spectral Methods
  - Dynamic programming
- Examples
- Summary

29/66

## Inventory of Key Computations




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

30/66

## CP1: Linear Algebra

- **Vector Space**: A set closed under + has identity and inverse elements, scalar multiplication
- **Linear Map**: Operator **T** on vectors u,v, scalar α s.t. **T**(u + v) = **T**u + **T**v, and **T**(αv) = α**T**(v)
- **Matrix**: An m × n array of numbers representing a Linear map from $R^n$ to $R^m$
- **Linear Equations**: Ax = b
- **Eigenvalues/vectors**: Ax = λx

31/66

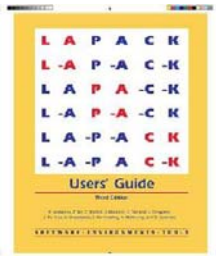
## Basic Linear Algebra Subroutines (BLAS)

- Three "Levels", known as BLAS, characterized by intrinsic ratio of computation to memory movement

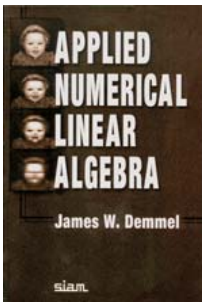
| Level | Example | # mem refs | # flops | q |
|---|---|---|---|---|
| 1 | xAXPY: y = y+αx | 3n | $2n^1$ | 2/3 |
| 2 | xGEMV: y=y+Ax | $n^2$ | $2n^2$ | 2 |
| 3 | xGEMM: C=C+AB | $4n^2$ | $2n^3$ | n/2 |

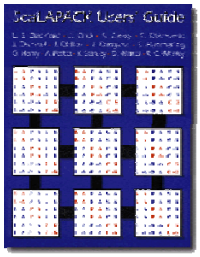
32/66

## Linear Algebra Resources

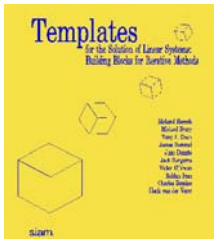



www.netlib.org/lapack




www.netlib.org/scalapack




www.netlib.org/templates

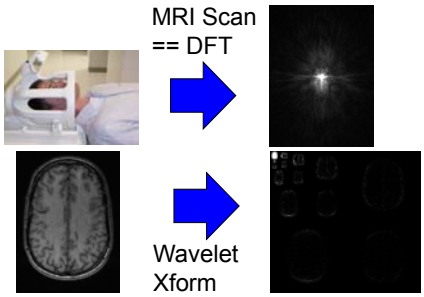
gams.nist.gov

www.cs.utk.edu/~dongarra/etemplates

33/66

## CP2: Spectral Methods Pattern: MRI Reconstruction

- "Spectral Methods" are a broad class of numerical algorithms for solving PDEs, but notions of Spectral Analysis (i.e. convenient changes of basis) are important in every application area
- In Magnetic Resonance Imaging (MRI), images are collected in "k-space" -- i.e. an MRI scan produces a Fourier Domain image
- Fourier and Wavelet representations are different Spectral analyses that expose different properties of images convenient for solving our problems




34/66

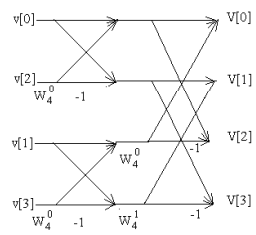
## Spectral Methods Pattern: Fast Transforms

- Spectral Methods rely on representions of data in "convenient" bases that produce working, computationally *feasible* algorithms
- Changing a basis is, in general, an $O(N^2)$ matrix-vector multiplication. The matrices representing "convenient" bases factor into O(N log N) fast transforms!

$$u_k = \sum_{j=0}^{N-1} x_j \, \omega^{jk}$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

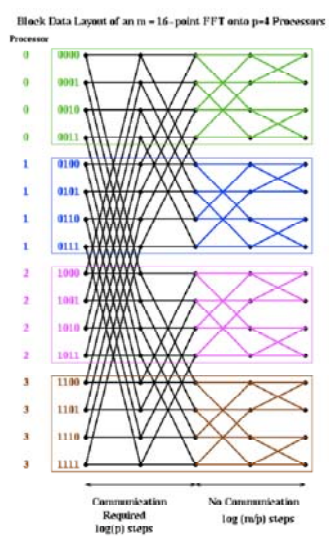
$$F_n \, x = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} \begin{bmatrix} F_{n/2} \, x_{even} \\ F_{n/2} \, x_{odd} \end{bmatrix}$$




35/66

## Spectral Methods Pattern: Libraries

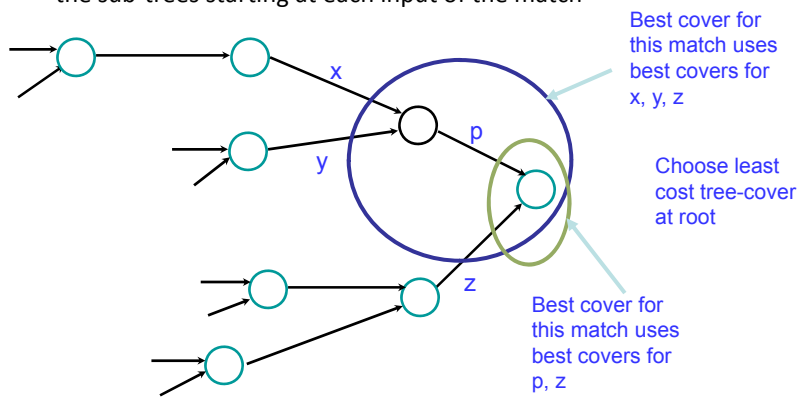
- Fast transform algorithms like the FFT are notoriously difficult to optimize:
- Luckily, implementations of the FFT exist for every platform. E.G:
  - FFTW and SPIRAL: Highly successful auto-tuners for FFT (and others) on PCs and workstations
  - CUFFT for Cuda on Nvidia GPUs




36/66

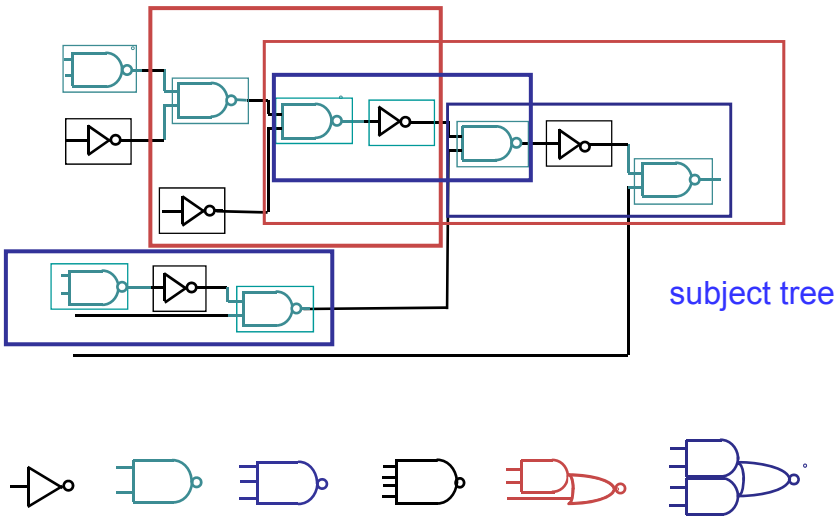
## CP3: Dynamic Programming

- Class of problems for which the optimal solution can be built up from optimal solutions to sub-problems
- Principle of optimality: Optimal cover for a tree consists of a best match at the root of the tree plus the optimal cover for the sub-trees starting at each input of the match



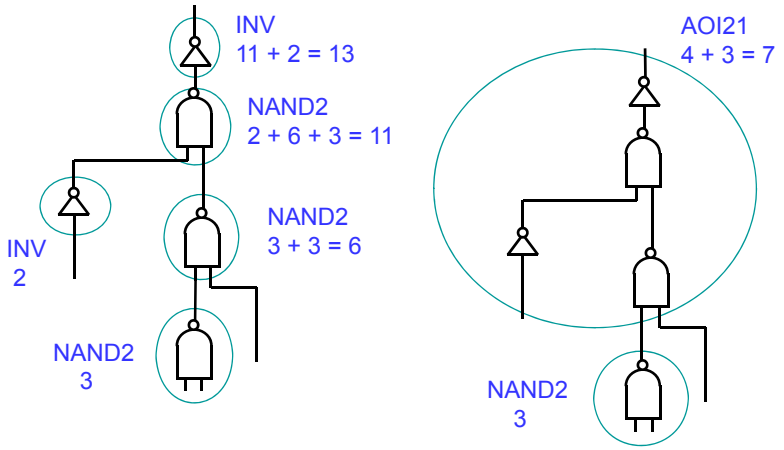

37/66

## Mapping a circuits into a Cell Library



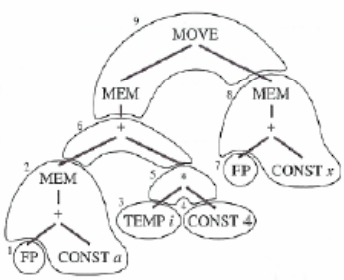

38/66

## Example of Optimal Tree Covering

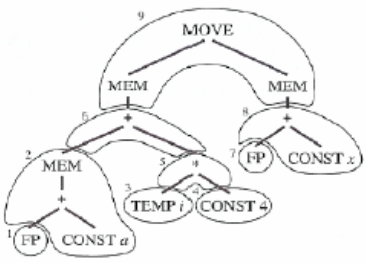



39/66

## code generation in compilers

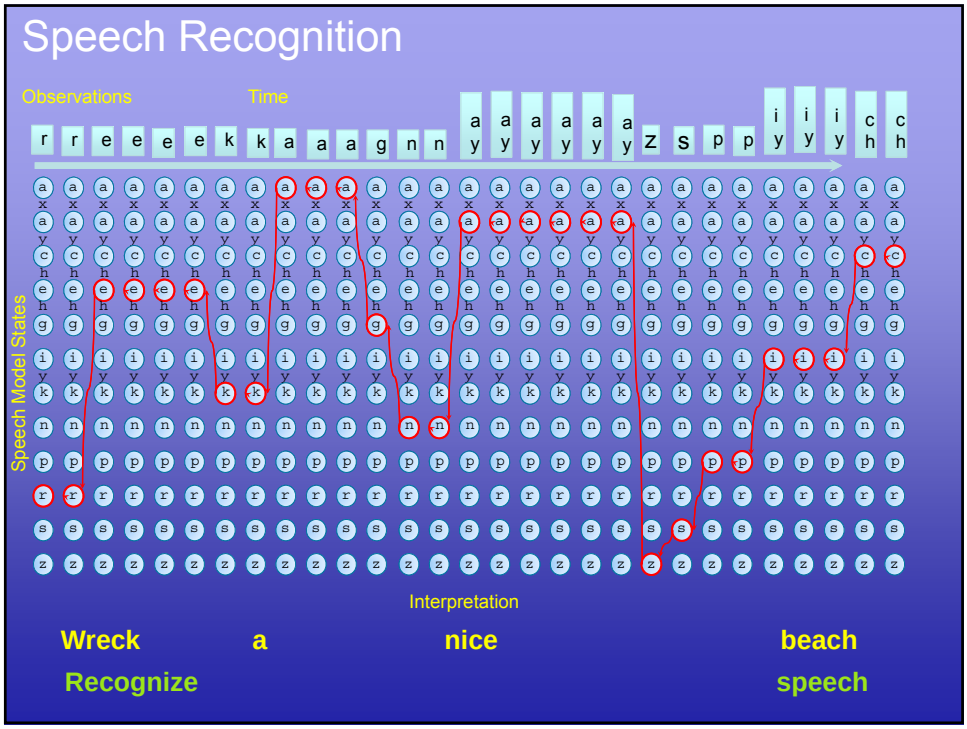



| | | |
|---|---|---|
| 2 | LOAD | $r_1 \leftarrow M[\mathbf{fp} + a]$ |
| 4 | ADDI | $r_2 \leftarrow r_0 + 4$ |
| 5 | MUL | $r_2 \leftarrow r_i \times r_2$ |
| 6 | ADD | $r_1 \leftarrow r_1 + r_2$ |
| 8 | LOAD | $r_2 \leftarrow M[\mathbf{fp} + x]$ |
| 9 | STORE | $M[r_1 + 0] \leftarrow r_2$ |




| | | |
|---|---|---|
| 2 | LOAD | $r_1 \leftarrow M[\mathbf{fp} + a]$ |
| 4 | ADDI | $r_2 \leftarrow r_0 + 4$ |
| 5 | MUL | $r_2 \leftarrow r_i \times r_2$ |
| 6 | ADD | $r_1 \leftarrow r_1 + r_2$ |
| 8 | ADDI | $r_2 \leftarrow \mathbf{fp} + x$ |
| 9 | MOVEM | $M[r_1] \leftarrow M[r_2]$ |

40/66



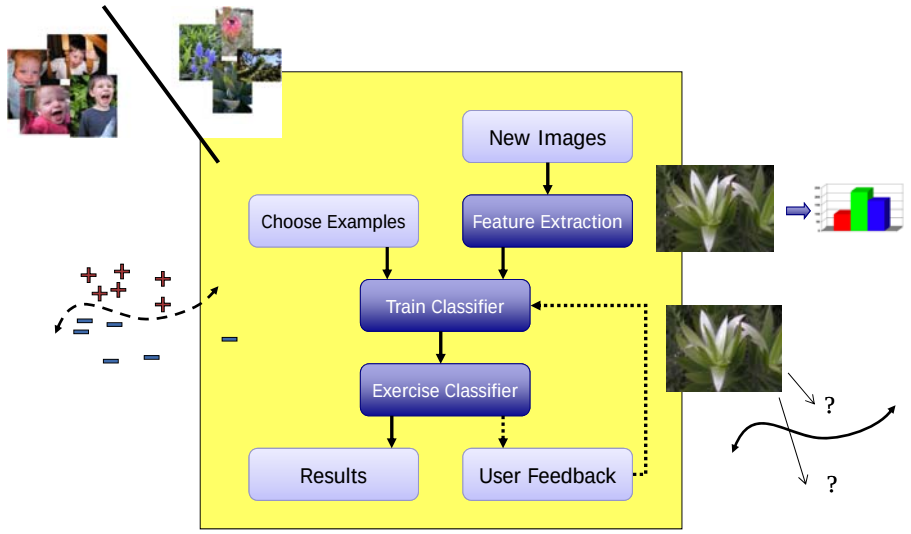

## Outline

PAMAS

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
  - Classification using Support Vector Machines
  - Speech Recognition
- Summary

42/66

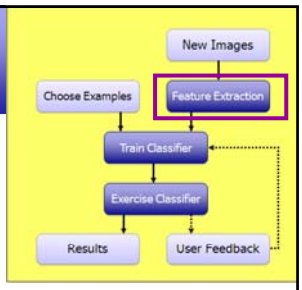
## Support Vector Machine Classifier



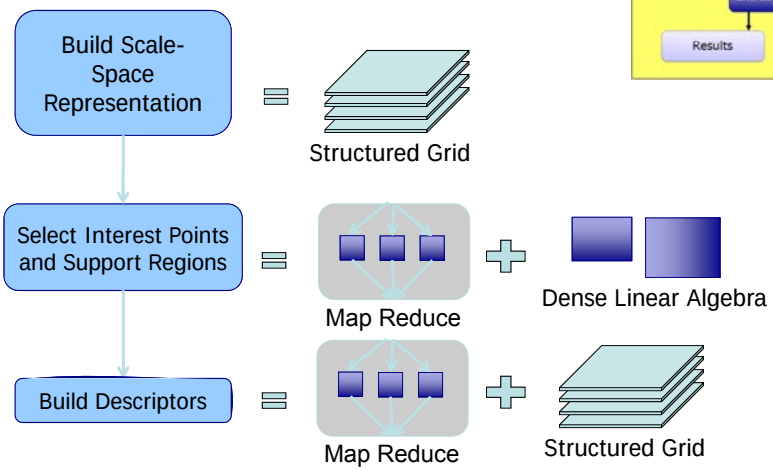

Catanzaro, Sundaram, Keutzer, "Fast SVM Training and Classification on Graphics Processors", ICML 2008

43/66

## Feature Extraction

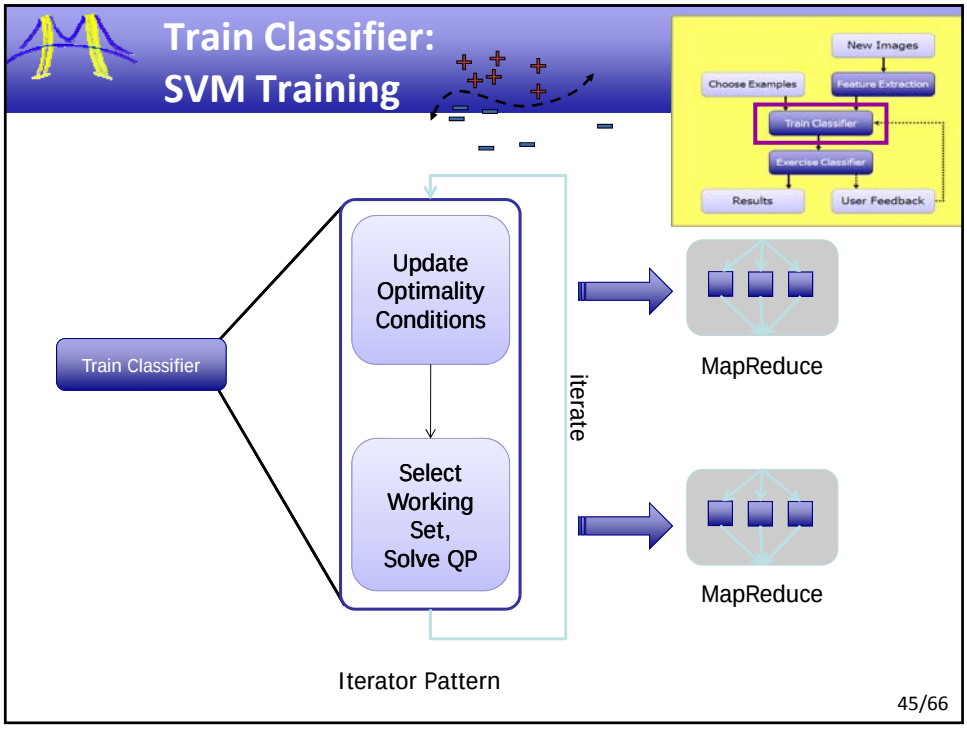


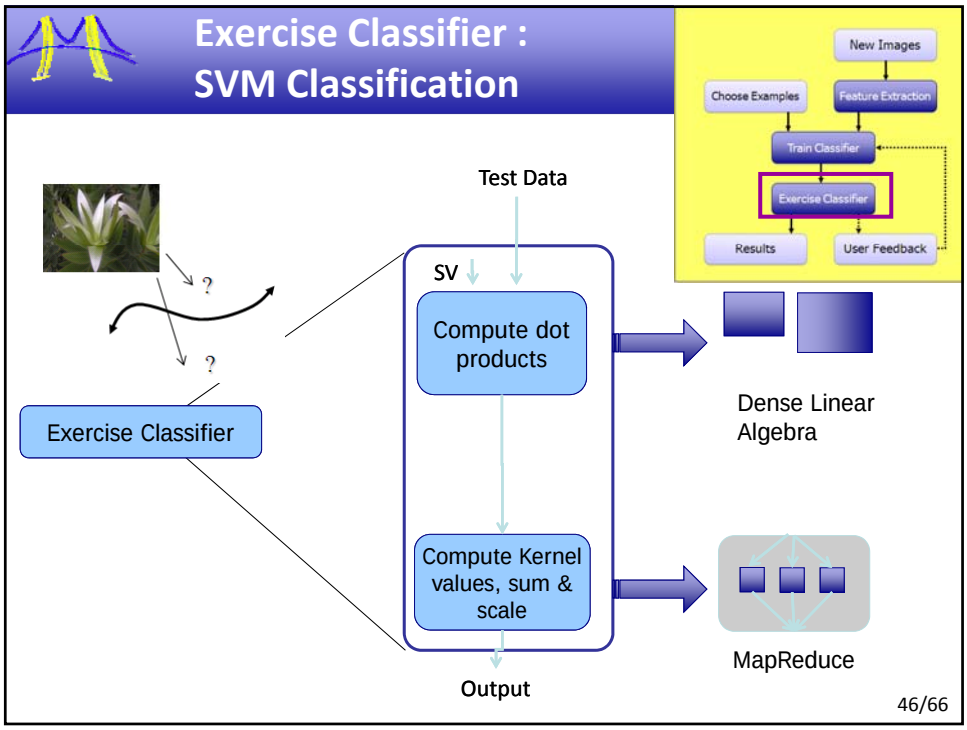

- Image is reduced to a set of low-dimensional feature vectors



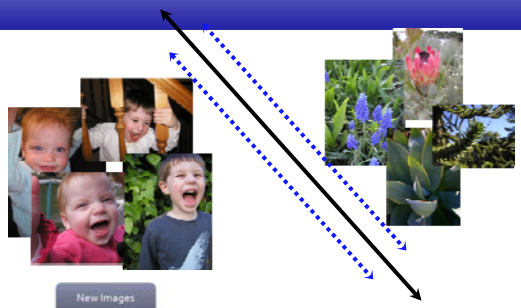

"Image Feature Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC '09

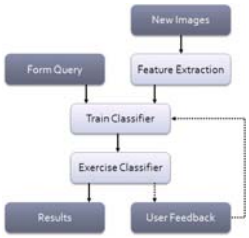
44/66


Train Classifier:
SVM Training
New Images
Choose Examples
Feature Extraction
Train Classifier
Exercise Classifier
Results
User Feedback
Train Classifier
Update Optimality Conditions
Select Working Set, Solve QP
iterate
MapReduce
MapReduce
Iterator Pattern
45/66


Exercise Classifier :
SVM Classification
New Images
Choose Examples
Feature Extraction
Train Classifier
Exercise Classifier
Results
User Feedback
Test Data
SV
Compute dot products
Compute Kernel values, sum & scale
Output
Exercise Classifier
Dense Linear Algebra
MapReduce
46/66

## Support-Vector Machine Mini-Framework






- Support-Vector Machine Framework used to achieve:
  - 9-35x speedup for training
  - 81-138x for classification
- 2200 downloads since release

Fast support vector machine training and classification , Catanzaro, Sundaram, Keutzer, International Conference on Machine Learning 2008

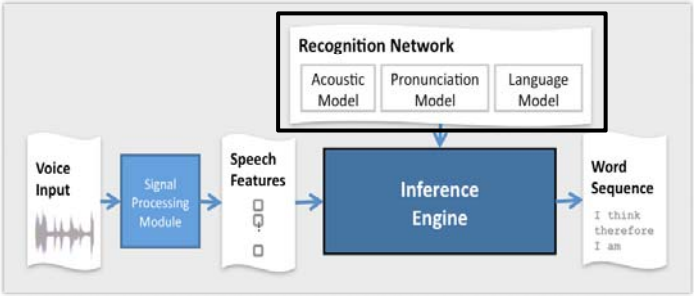
47/66

## Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
  - CBIR
  - Speech Recognition
- Summary

48/66

## Speech Recognition System Structure




- 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

49/66

## Architecting Parallel Software

Decompose Tasks
- Group tasks
- Order Tasks

Decompose Data
- Data sharing
- Data access

Identify the Software Structure → Identify the Key Computations

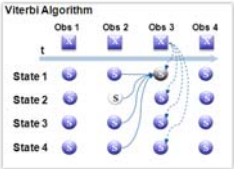
50/66

## Inference Engine Architecture

Intro

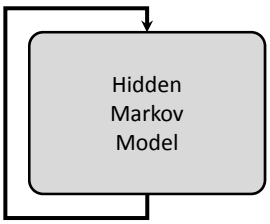
- Recognition is a process of graph traversal/dynamic programming
- Each time-step we need to identify the likely states in the recognition network given the observation acoustic signal
- From a the of active states we want to compute the next set of active states using probabilities of acoustic symbols and state transitions
- What **Structural pattern** is this?

| 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 | |




51/66

## 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 | |

52/66

## Key computation: HMM Inference Algorithm

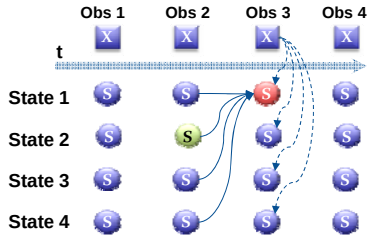
An instance of: **Graphical Models**

Implemented with: **Dynamic Programming**

- Finds the most-likely sequence of states that produced the observation

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

**Viterbi Algorithm**




**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:**

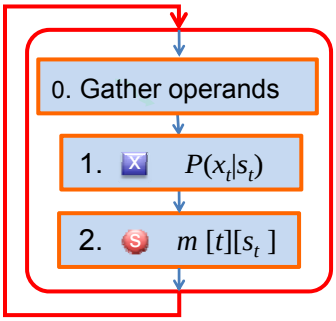
$$m[t][s_t] \doteq \max_{s_0,\ldots,s_{t-1}} P(x_0,\ldots,x_t,s_0,\ldots,s_{t-1},s_t)$$

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

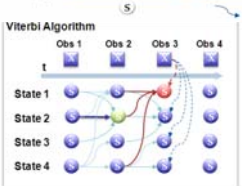
53/66

## Inference Engine in LVCSR

- Structural pattern to support three phases of inference
  0. Gather operands from irregular data structure to runtime buffer
  1. Perform observation probability computation
  2. Perform graph traversal computation




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




| | |
|---|---|
| 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 | |

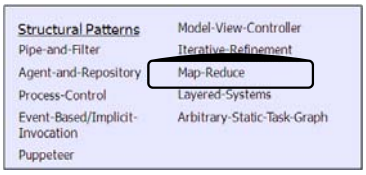
54/66

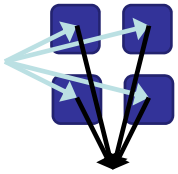
## Each Filter is a Map Reduce
## 0. Gather operands

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

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

0. Gather operand



**Coalesced data**

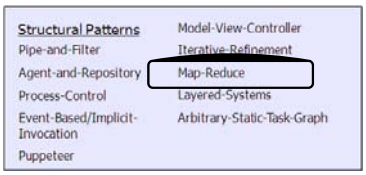
55/66

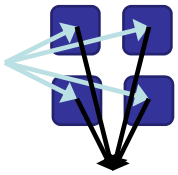
## Each Filter is Map Reduce
## 1. observation probability computation

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

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

1. $P(x_t|s_t)$



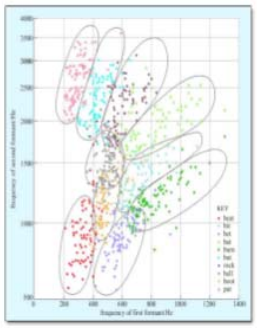



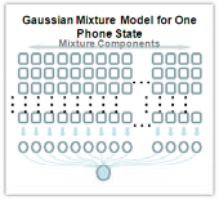
**GMM probability**

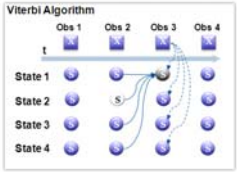
56/66

## Phase 1: Observation Probability computation Architecture

- 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





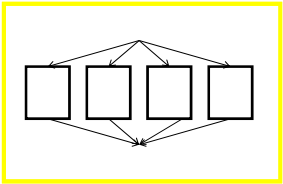




| Structural Patterns | |
|---|---|
| Pipe-and-Filter | Model-View-Controller |
| Agent-and-Repository | Iterative-Refinement |
| Process-Control | Map-Reduce |
| Event-Based/Implicit-Invocation | Layered-Systems |
| Puppeteer | Arbitrary-Static-Task-Graph |

Dan Klein's CS288, Lecture 9

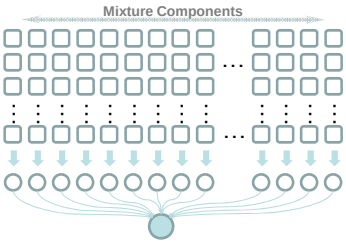
57/66

## Map-Reduce Structural Pattern



- Map each mixture probability computation
- Reduce the result – accumulate the total probability for that state

Gaussian Mixture Model for One Phone State




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

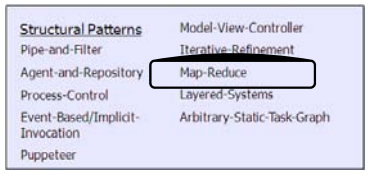
58/66

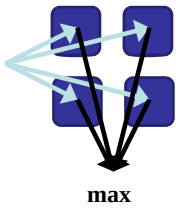
## Each Filter is Map Reduce
## 2. graph traversal computation

PAMAS

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

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



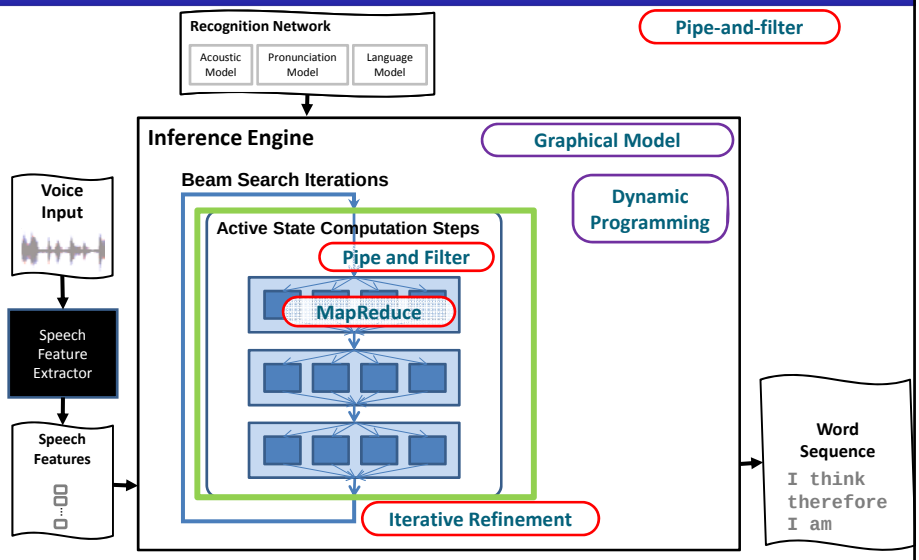




2. $m[t][s_t]$

59/66

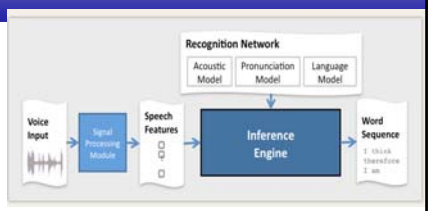
## LVCSR Software Architecture

PAMAS




60/66

## Speech Recognition Results




- Input: Speech audio waveform
- Output: Recognized word sequences

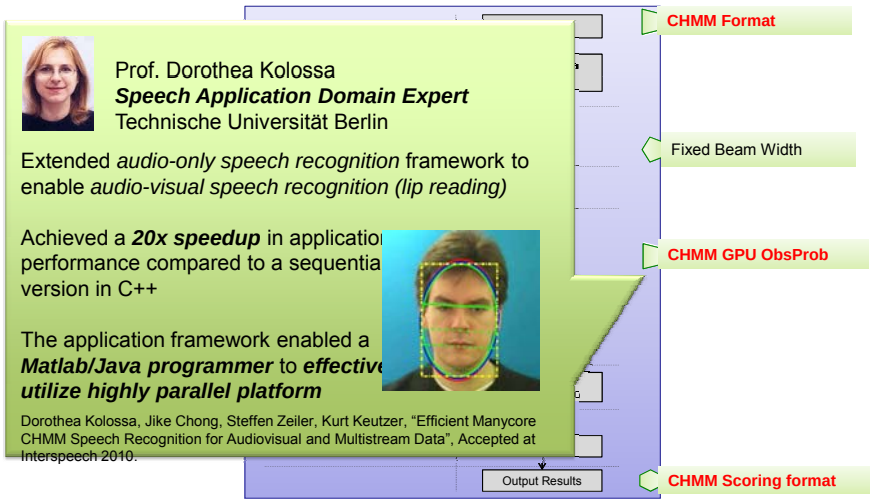
- Achieved 11x speedup over sequential version
- Allows 3.5x faster than real time recognition

- Our technique is being deployed in a hotline call-center data analytics company
- Used to search content, track service quality and provide early detection of service issues

Scalable HMM based Inference Engine in Large Vocabulary Continuous Speech Recognition, Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher Hughes, Wonyong Sung and Kurt Keutzer, IEEE Signal Processing Magazine, March 2010

61/66

## Multi-media Speech Recognition




Prof. Dorothea Kolossa
***Speech Application Domain Expert***
Technische Universität Berlin

Extended *audio-only speech recognition* framework to enable *audio-visual speech recognition (lip reading)*

Achieved a ***20x speedup*** in applicatio performance compared to a sequentia version in C++

The application framework enabled a ***Matlab/Java programmer*** to ***effectiv utilize highly parallel platform***

Dorothea Kolossa, Jike Chong, Steffen Zeiler, Kurt Keutzer, "Efficient Manycore CHMM Speech Recognition for Audiovisual and Multistream Data", Accepted at Interspeech 2010.

62/66

## Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
- Summary

63/66

## Other Interesting Results

- Patterns have helped the PALLAS research group publish papers in a diverse group of leading Computer Science conferences in the last few years:
  - Interspeech 2009, Interspeech 2010 (2)
  - IEEE Signal Processing Magazine 2009
  - European Conference on Computer Vision 2010
  - International Conference on Computer Vision 2009
  - International Parallel and Distributed Processing Symposium 2012, 2011, 2009 (2)
  - Workshop on High Performance Computing in Finance at Super Computing 2009
  - Joint Annual Meeting of the International Society for Magnetic Resonance in Medicine, *ISMRM 2010*
  - International Conference on Machine Learning 2008
  - IEEE Transactions on Medical Imaging (2012)
- What's the point?
  - Computational patterns give a new powerful viewpoint to efficiency programmers:
  - Enable us to disentangle the big fuzzy ball of yarn of computation
    - add 20 IQ points to our problem solving (as per Alan Kay)
  - Our Pattern language helps you to write good parallel code

64/66

## Speedups

| Application | Speedups |
|---|---|
| MRI | 100x |
| SVM-train | 20x |
| SVM-classify | 109x |
| Contour | 130x |
| Object Recognition | 80x |
| Poselet | 20x |
| Optical Flow | 32x |
| Speech | 11x |
| Value-at-risk | 60x |
| Option Pricing | 25x |

65/66

## Summary

- 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 three –seven more will be all you need
- Computational patterns: provide *efficiency,* identify key computations to be parallelized
  - I showed you three –ten more will be all you need
- Orchestration of computational and structural patterns creates architectures which greatly facilitates the development of parallel programs:
  - I showed you three – there are *many* more

Patterns: http://parlab.eecs.berkeley.edu/wiki/patterns/patterns

PALLAS: http://parlab.eecs.berkeley.edu/research/pallas

66/66

## Extras

PAMAS

67/66