Microsoft

Microsoft® Research Faculty Summit 2012

ADVANCING THE STATE OF THE ART

Evolution of Parallel Patterns: from Design tool to development tool

Tim Mattson
Intel Labs

Disclaimer

READ THIS … its very important

- The views expressed in this talk are those of the speaker and not his employer.
- This is an academic style talk and does not address details of any particular Intel product. You will learn nothing about Intel products from this presentation.
- This was a team effort, but if I say anything really stupid, it's my fault … don't blame my collaborators.

Slides marked with this symbol were produced by Kurt Keutzer and myself for CS194 … A UC Berkeley course on Architecting parallel applications with Design Patterns.

The many core challenge

- A harsh assessment …
  - We have turned to multi-core chips not because of the success of our parallel software but because of our failure to continually increase CPU frequency.

- Result: a fundamental and dangerous mismatch
  - Parallel hardware is ubiquitous … Parallel software is rare

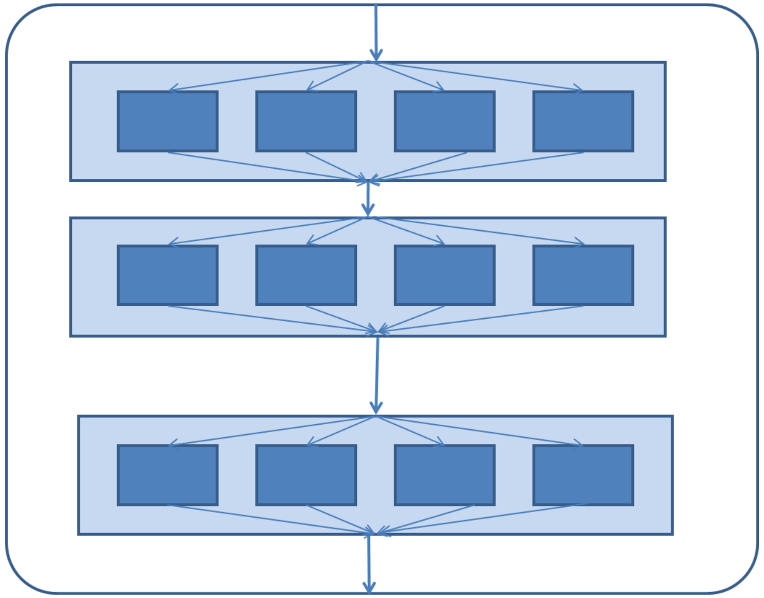
- The Many Core challenge …
  - Parallel software must become as common as parallel hardware

After ~30 years of parallel computing research, we know:
(1) automatic parallelism doesn't work
(2) an endless quest for the perfect parallel language is counterproductive …
"worse is better" **(Richard Gabriel, 1991)**

So how can we address the many core challenge?

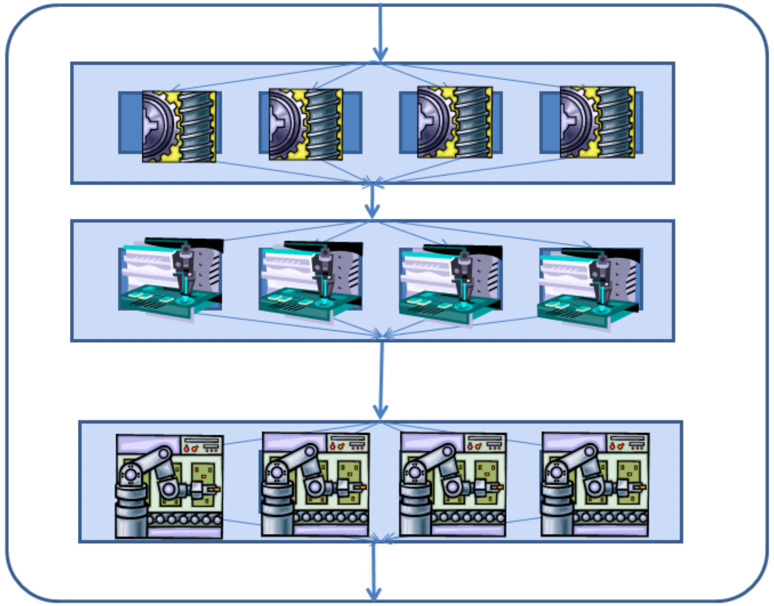
Architecting Parallel Software

- We believe the solution to parallel programming starts with developing a good software architecture



Architecting Parallel Software

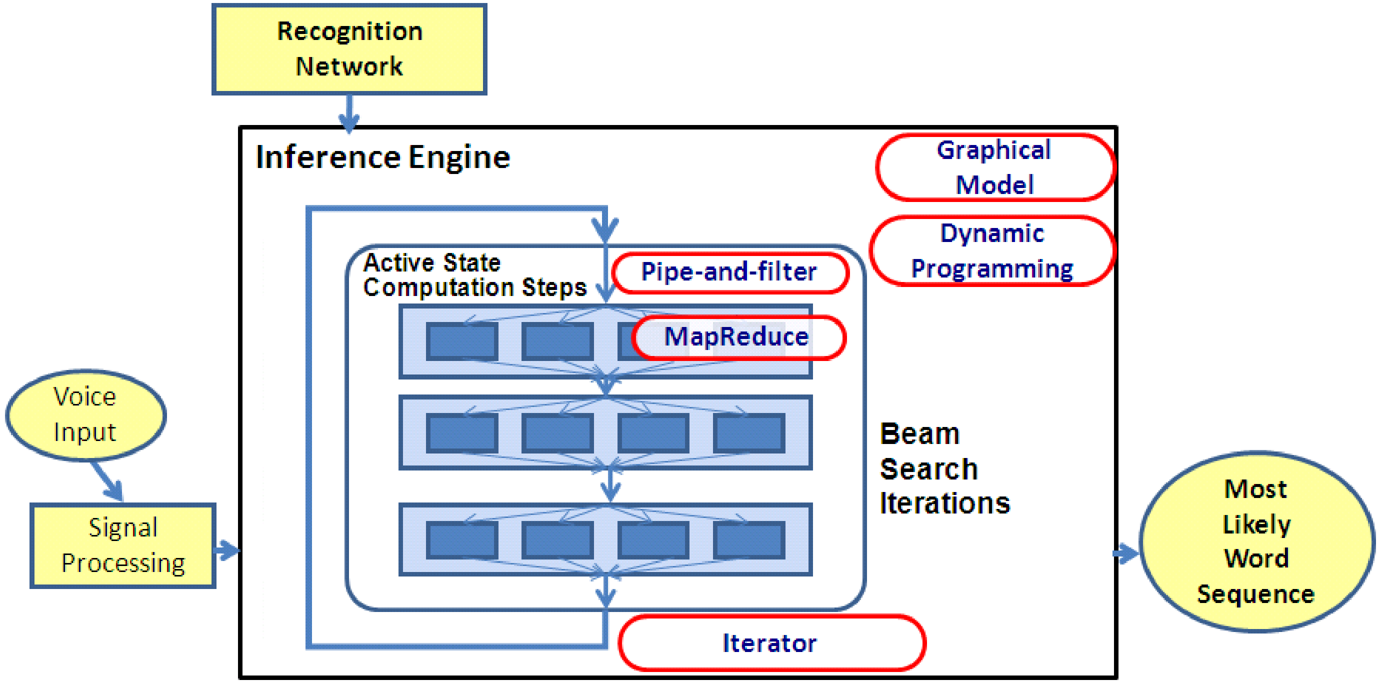
- We believe the solution to parallel programming starts with developing a good software architecture



- Analogy: the layout of machines/processes in a factory

Architecting Parallel Software

- We believe the solution to parallel programming starts with developing a good software architecture



- Example: SW Architecture of Large-Vocabulary Continuous Speech Recognition

… and how do we systematically describe software architectures?

Our Pattern Language (OPL 2012)

Applications

Structural Patterns

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

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

Finding Concurrency Patterns

- Task Decomposition
- Data Decomposition
- Ordered task groups
- Data sharing
- Design Evaluation

Parallel Algorithm Strategy Patterns

- Task-Parallelism
- Divide and Conquer
- Data-Parallelism
- Pipeline
- Discrete-Event
- Geometric-Decomposition
- Speculation

Implementation Strategy Patterns

Program structure

- SPMD
- Kernel-Par.
- Fork/Join
- Actors
- Vector-Par
- Loop-Par.
- Workpile

Algorithms and Data structure

- Shared-Queue
- Shared-Map
- Parallel Graph Traversal
- Distributed-Array
- Shared-Data

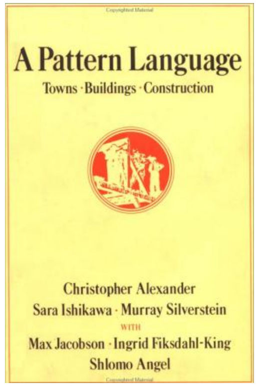
Parallel Execution Patterns

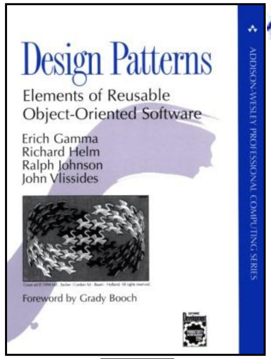
- Coordinating Processes
- Stream processing
- Shared Address Space Threads
- Task Driven Execution

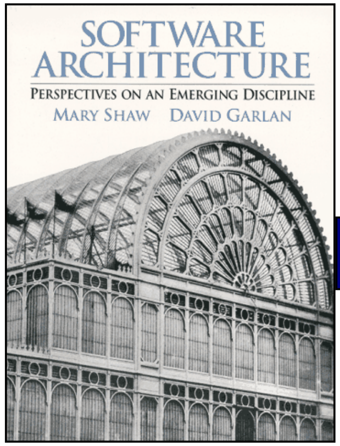
Concurrency Foundation constructs (not expressed as patterns)

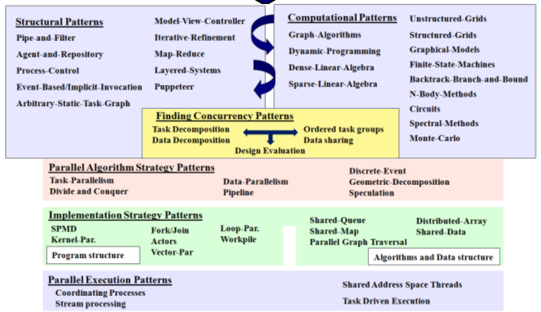
- Thread/proc management
- Communication
- Synchronization

Researchers from UCB, Intel, UIUC, and others collaborated to create "the grand canonical pattern language" of parallel application programming.

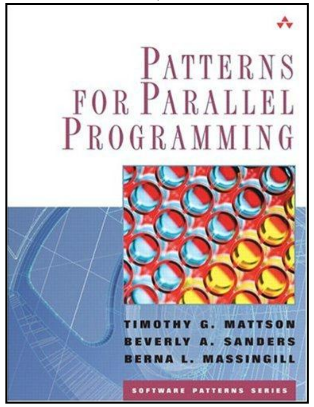




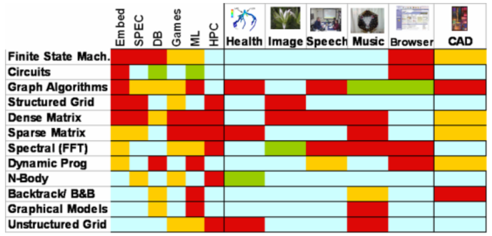




"Our Pattern Language" (OPL)

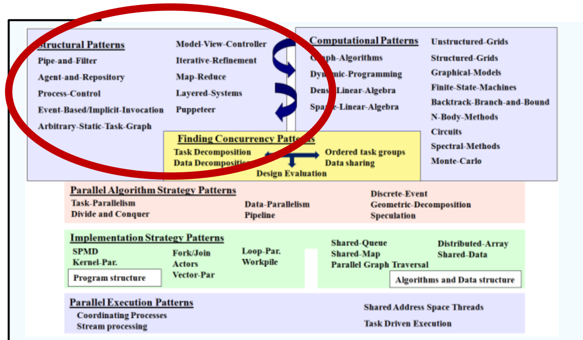


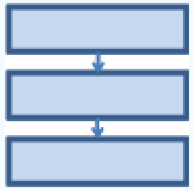
Pattern Language of Parallel Programming (PLPP)

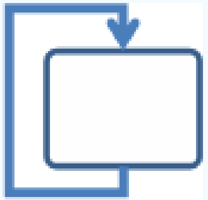


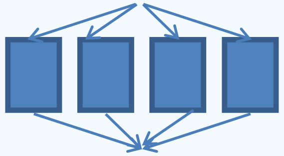
13 dwarves

Pattern examples



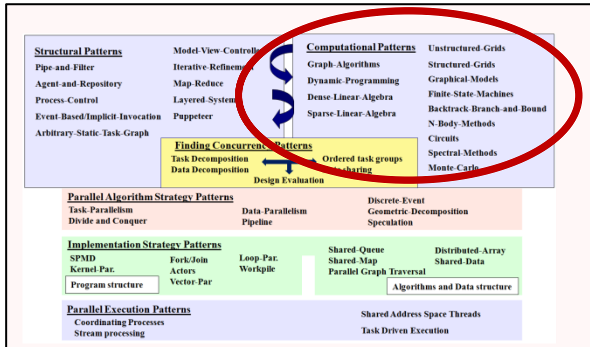


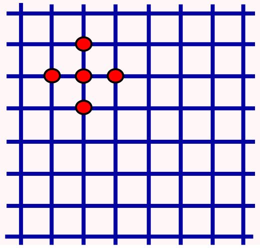


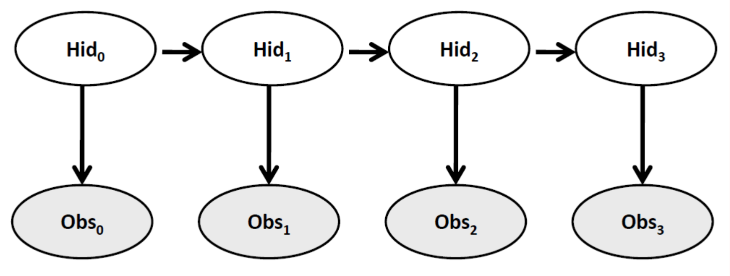


•Pipe-and-Filter •Iterative refinement •MapReduce

Structural Patterns: Define the software structure .. *Not* what is computed

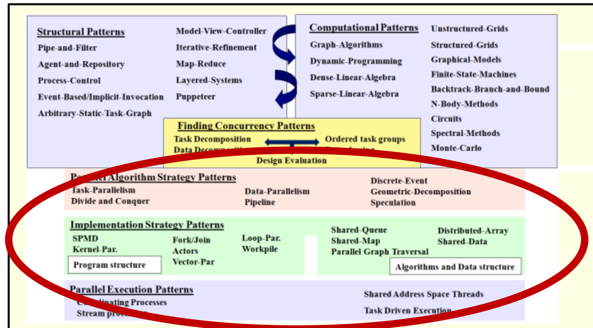






•Structured mesh •Graphical Models

Computational Patterns: Define the computations "inside the boxes"



•Fork-join •SPMD •Data parallel

Parallel Patterns: Defines parallel algorithms

OPL Pattern Language

Applications

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

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

Patterns travel together … informs framework design (a pathway for cactus is shown here)

Patterns

Ordered task groups
Data sharing
gn Evaluation

Parallel Algorithm Strategy Patterns
- Task-Parallelism
- Divide and Conquer
- Data-Parallelism
- Pipeline
- Discrete-Event
- Geometric-Decomposition
- Speculation

Implementation Strategy Patterns
- SPMD
- Kernel-Par.
- Fork/Join
- Actors
- Vector-Par
- Loop-Par.
- Workpile
- Shared-Queue
- Shared-Map
- Parallel Graph Traversal
- Distributed-Array
- Shared-Data

Program structure

Algorithms and Data structure

Parallel Execution Patterns
- Coordinating Processes
- Stream Processing
- Shared Address Space Threads
- Task Driven Execution

Distributed memory cluster and MPP computers

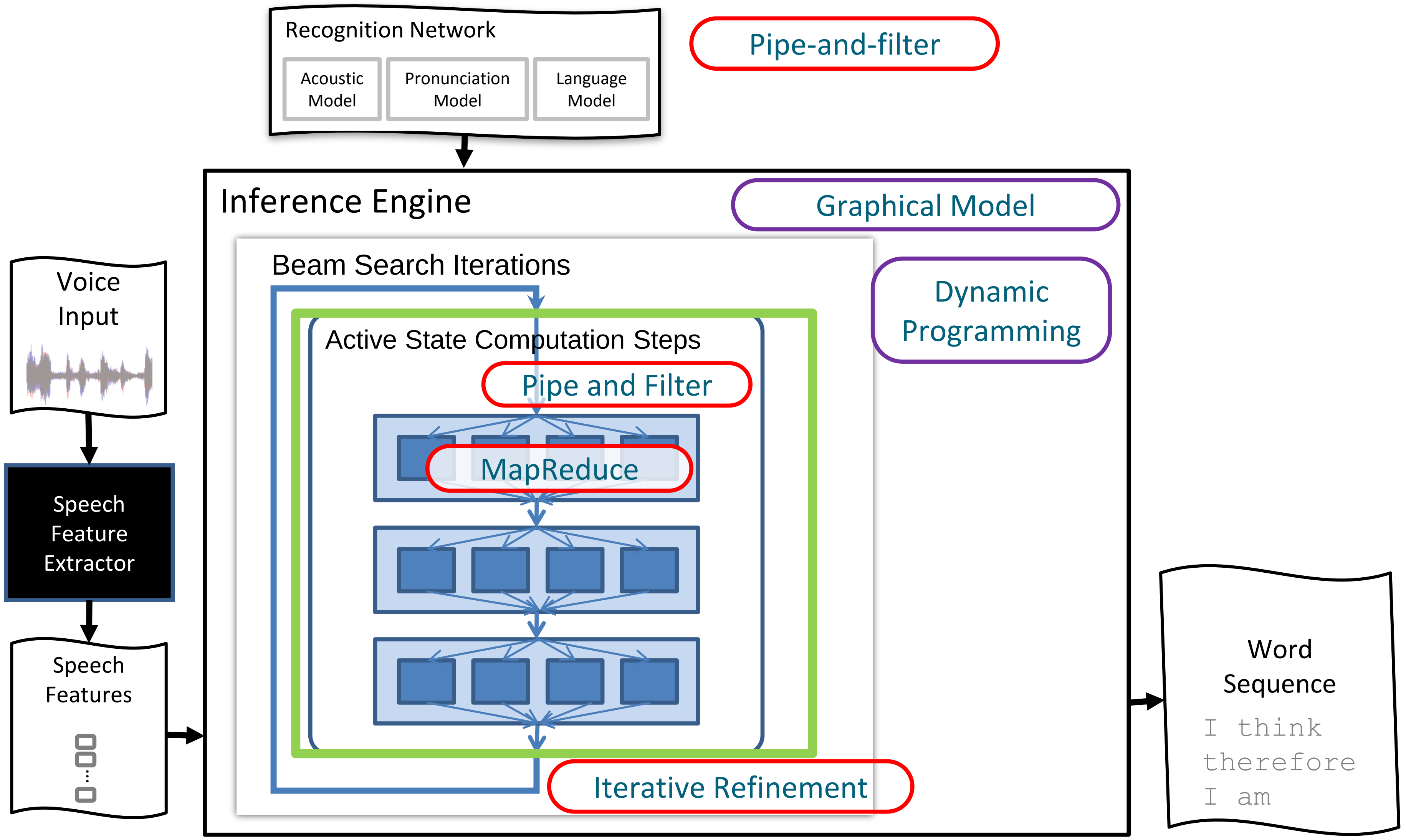
expressed as patterns)

Commu

Multiprocessors (SMP and NUMA)

Source: Keutzer and Mattson Intel Technology Journal, 2010

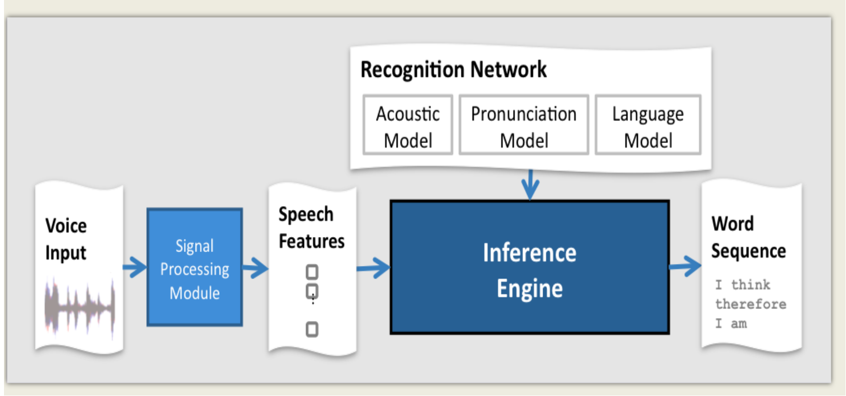
LVCSR Software Architecture



LVCSR = Large vocabulary continuous speech recognition.

Speech Recognition Results

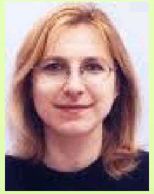
- Architecture expressed as a composition of design patterns and implemented as a C++ Framework.
  - Input: Speech audio waveform
  - Output: Recognized word sequences



- 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

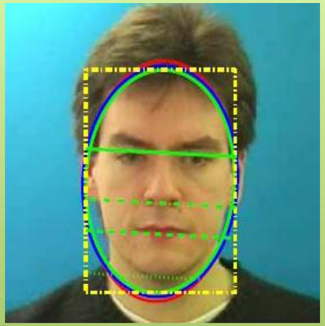
Scalable HMM based Inference Engine in Large Vocabulary Continuous Speech Recognition, Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher Hughes, Wonyong Sung and Kurt Keutzer, IEEE Signal Processing Magazine, March 2010

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 application performance compared to a sequential version in C++

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

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

Output Results

CHMM Format

Fixed Beam Width

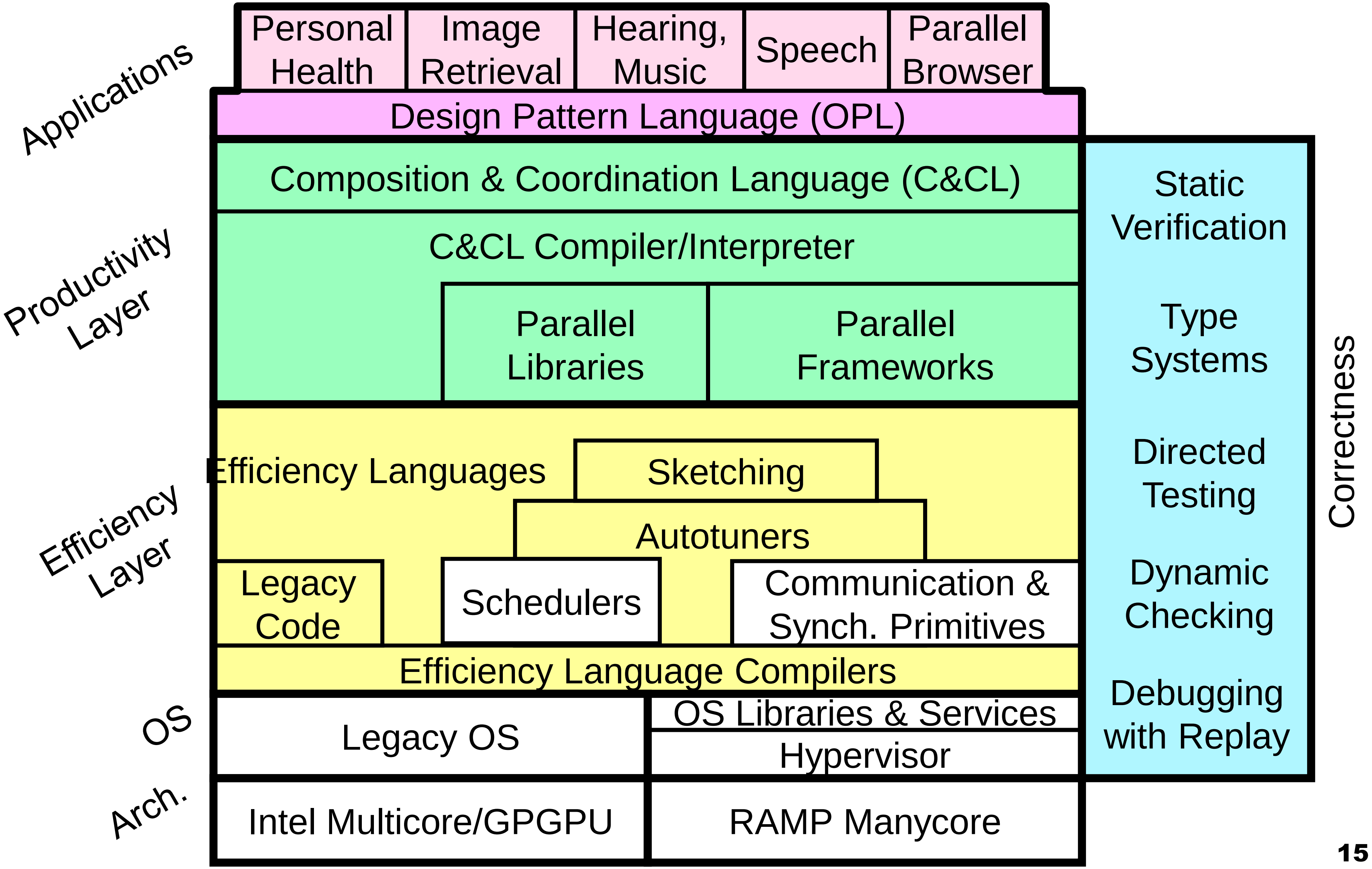
CHMM GPU ObsProb

CHMM Scoring format

Source: K. Keutzer and his research group at UCB, slides from CS194 Spring 2012

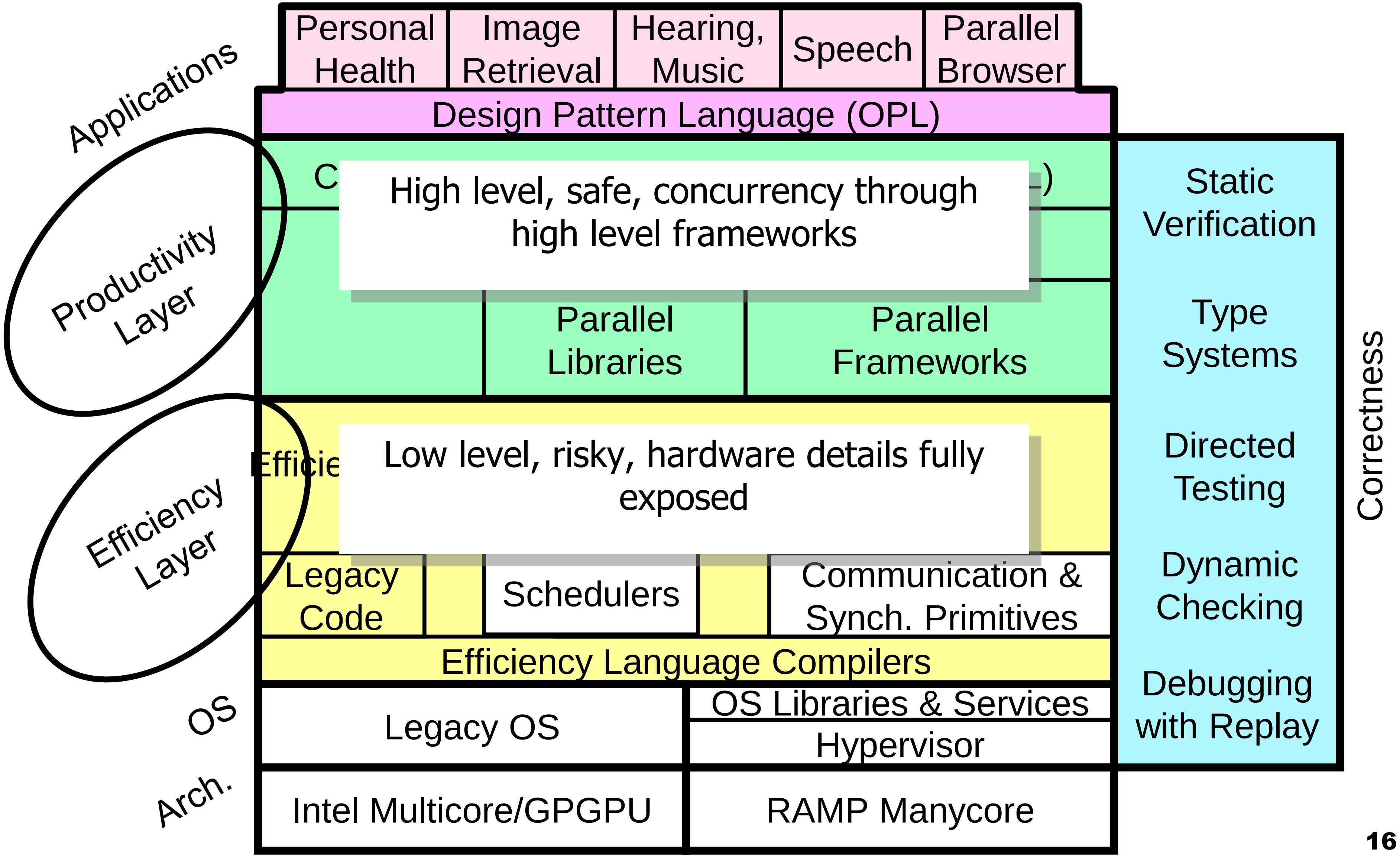
Par Lab Research Overview

Easy to write correct software that runs efficiently on manycore



Par Lab Research Overview

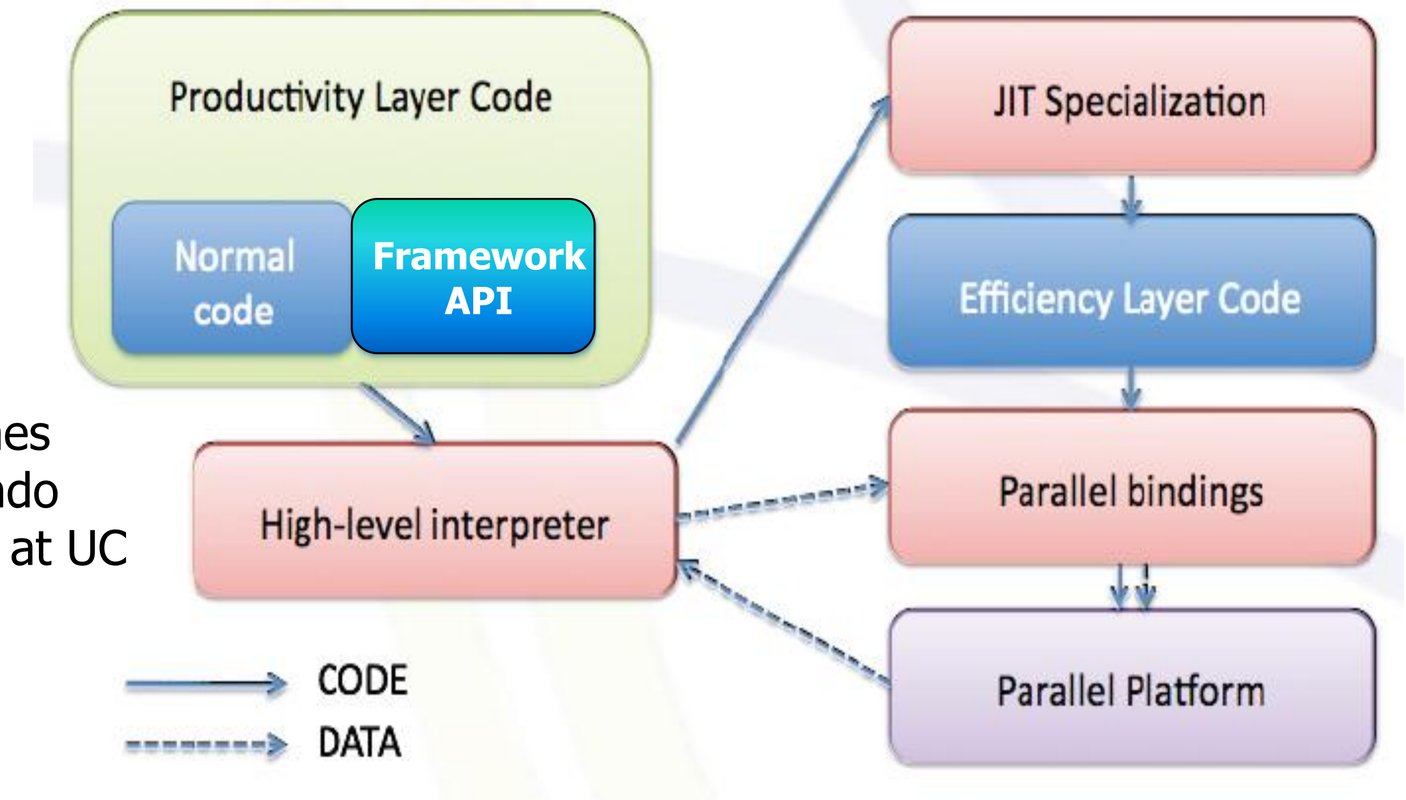
Easy to write correct software that runs efficiently on manycore



How do we squeeze high performance from framework-based applications?

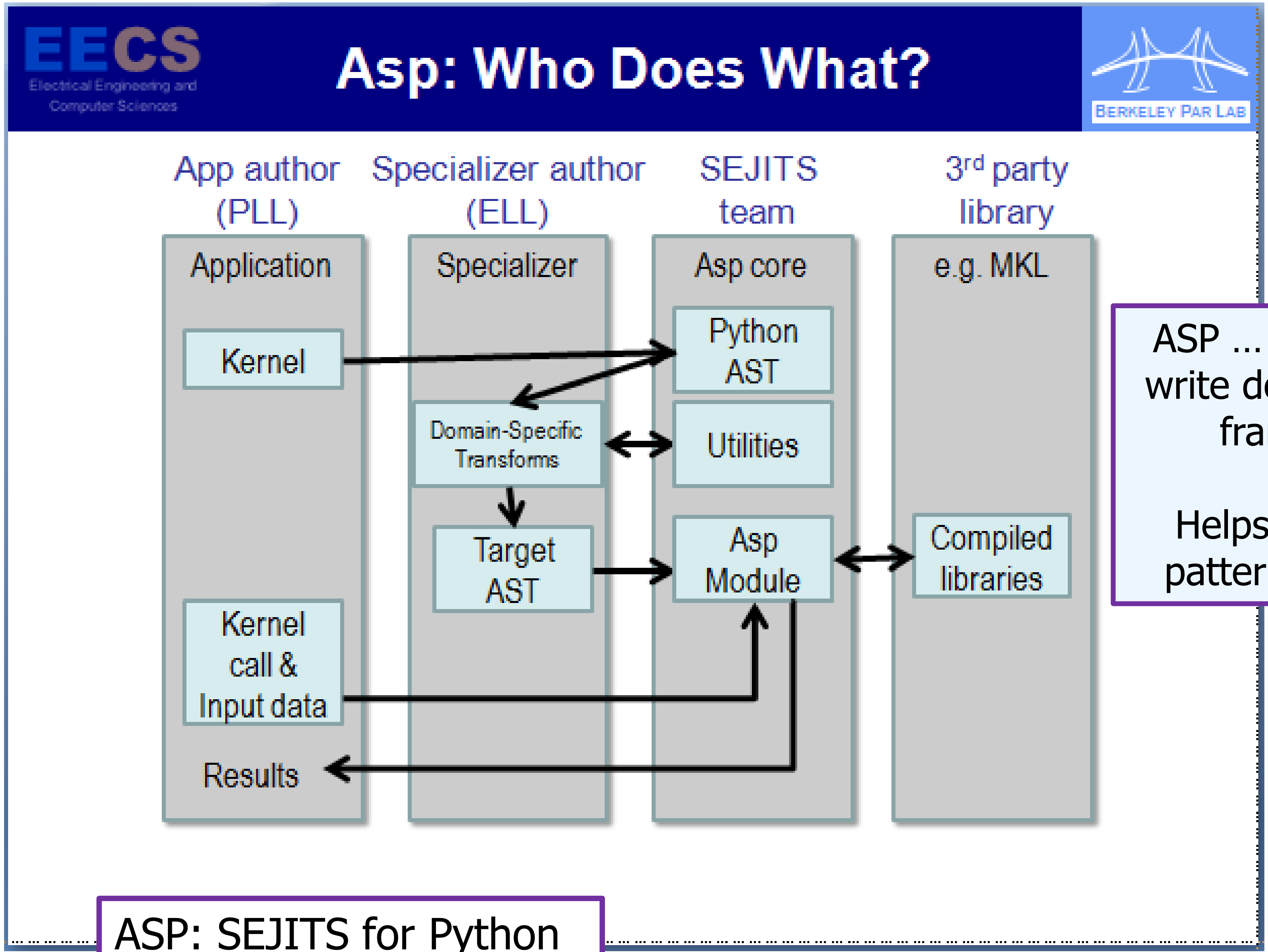
- SEJITS: Scalable, embedded, just in time specialization
  - Code with a high level language (e.g. Python or Ruby) that is mapped onto a low level, efficiency language (e.g. OpenMP/C or CUDA).
  - SEJITS system to embed optimized kernels specialized at runtime to flatten abstraction overhead and map onto hardware features.

SEJITS comes from Armando Fox's group at UC Berkeley.



Bryan Catanzaro, Armando fox, Yunsup Lee, mark Murphy and Kurt Ketuzer of UC Berkeley, Mickael Garland of NVIDIA

Turning Patterns expressed as Python code into high performance parallel code



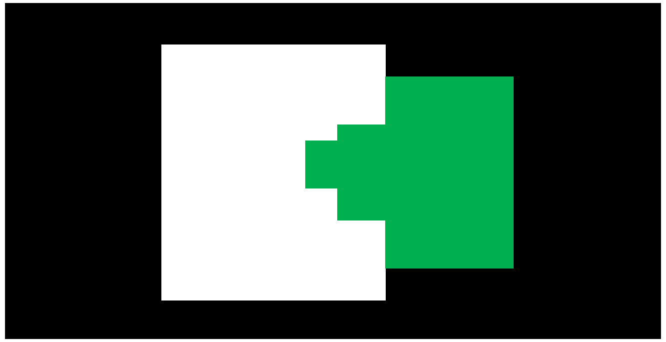
ASP … a platform to write domain specific frameworks.

Helps turn design patterns into code.

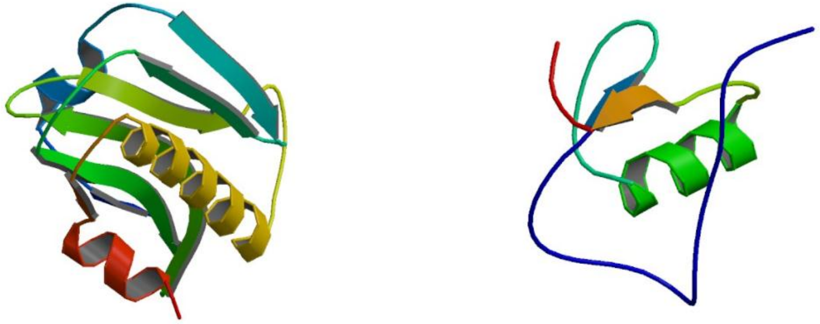
ASP: SEJITS for Python

Example Application: Shape Fitting

How do these two shapes fit together?



Pretty obvious.



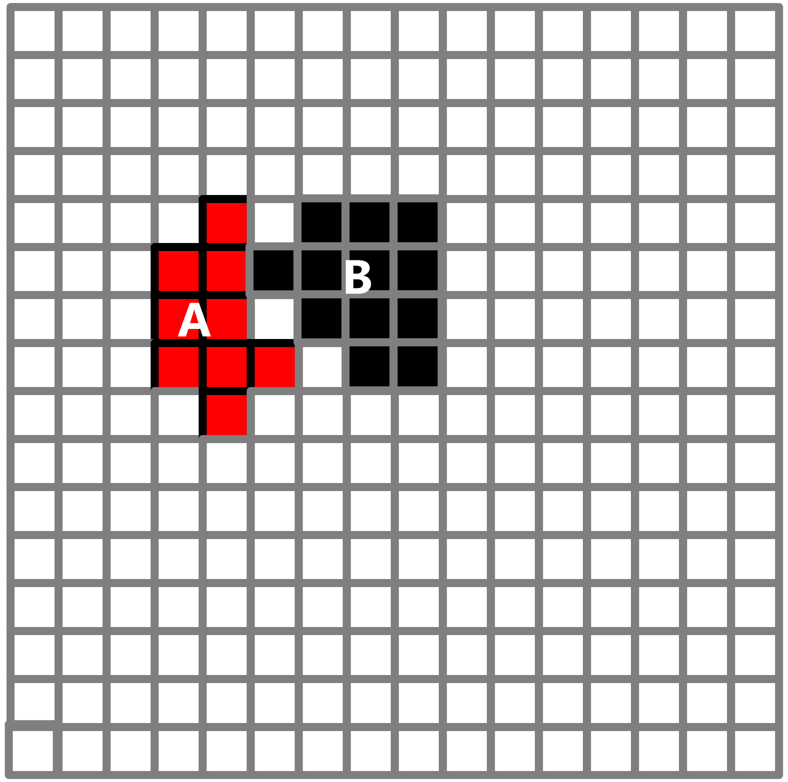
How do *these* two shapes fit together? Not as obvious when dealing with complex, 3D molecular structures.

Why does it matter how molecules fit together? Because most biological processes involve molecular binding.

Source: Henry Gabb, parlab retreat winter 2011

Shape Fitting by Cartesian Grid Correlations

Project molecules A and B onto a grid and assign values to nodes based on locations of atoms.



$$C_{\alpha\beta\gamma} = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} A_{ijk} \times B_{i+\alpha,j+\beta,k+\gamma}$$

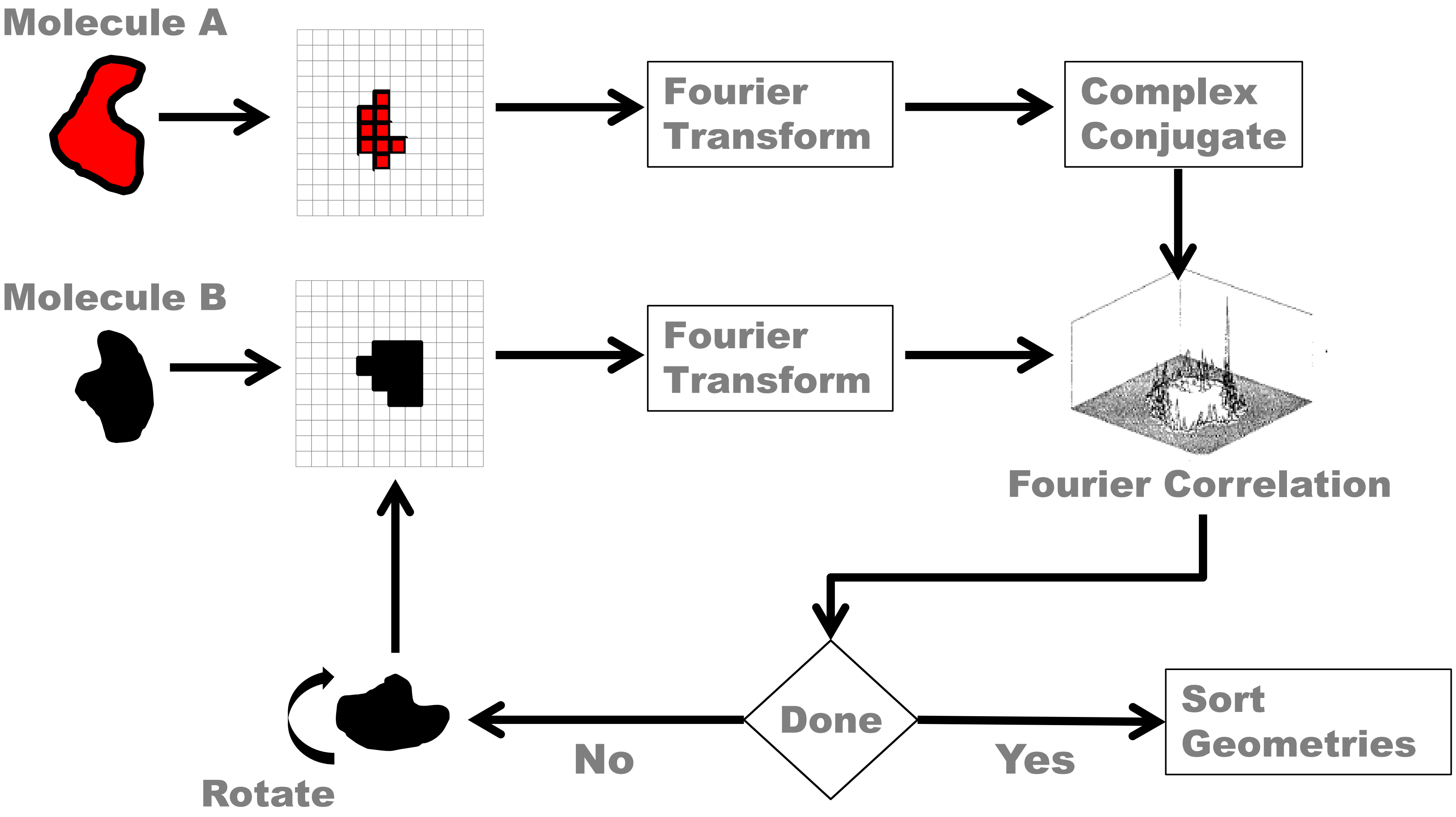
Translate/rotate molecules to maximize the correlation.

Inefficient: $O(N^6)$, N^3 additions and multiplications for every N^3 translations (α, β, γ).

Solve more efficiently using Fourier correlation: $O(N^3 \log N^3)$.

Source: Henry Gabb, parlab retreat winter 2011

Application “Box-and-Arrow” Diagram



Source: Henry Gabb, parlab retreat winter 2011

Productivity Programmer Responsibilities

Original loop-based, iterative code:

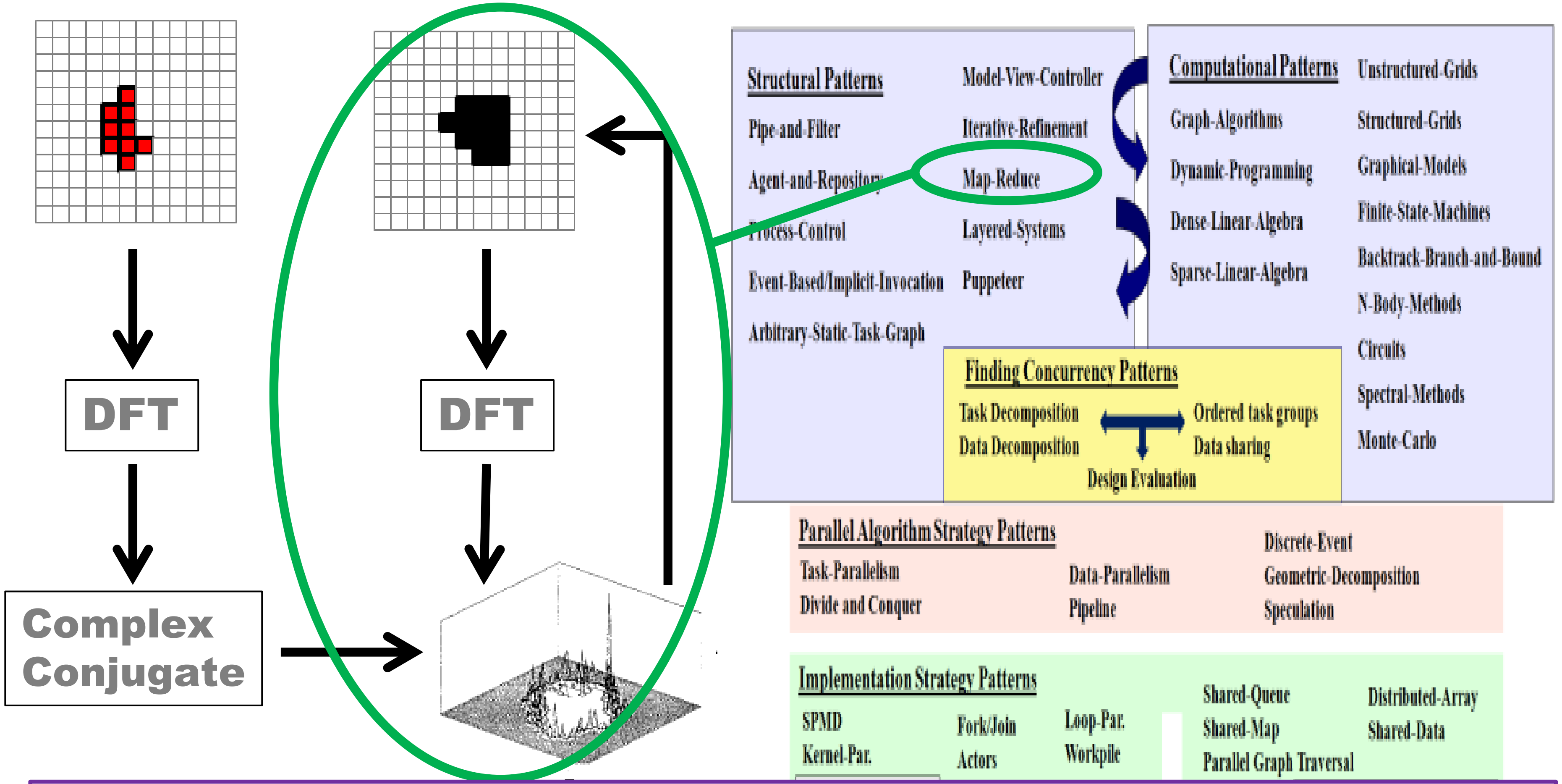
```
for a in range(-1.0, 1.0 + del, del):
   for b in range(-1.0, 1.0 + del, del):
      for g in range(-1.0, 1.0 + del, del):

         # ftdock algorithm
```

The productivity programmer knows the body of this loop-nest is "embarrassingly parallel" … but there is no way a compiler could figure this out

Source: Henry Gabb, parlab retreat winter 2011

Parallel Design Patterns



To expose the most concurrency in a natural way, it was best to recast the problem in terms of map-reduce.

i.e. the productivity programmer is responsible for a good design.

Source: Henry Gabb, parlab retreat winter 2011

Productivity Programmer Responsibilities

Original loop-based, iterative code:

```
for a in range(-1.0, 1.0 + del, del):
   for b in range(-1.0, 1.0 + del, del):
      for g in range(-1.0, 1.0 + del, del):

         # ftdock algorithm
```

New Code inspired by the map-reduce pattern:

```
a = b = g = list(range(-1.0, 1.0 + del, del))
geometries = AllCombMap([a, b, g], ftdock, *args)
```

Source: Henry Gabb, parlab retreat winter 2011

SEJITS/FTDock Results

- What SEJITS did for FTDock
  - Parallelism exploited though a map-reduce module
  - Mapped FFTW onto the application … with no changes to application code.
- Minimal burden on productivity programmer:
  - Pattern-based design of application
  - Functional programming style
  - Significantly easier development:
  - Original version: 4,700 lines of C and Perl
  - New version: 500 lines of Python
    - *Caveat: LOC not necessarily a good measure of productivity*
- Performance (16-core Xeon):
  - Serial: ~24 hours
  - Parallel: ~3 hours

Source: Henry Gabb, parlab retreat winter 2011

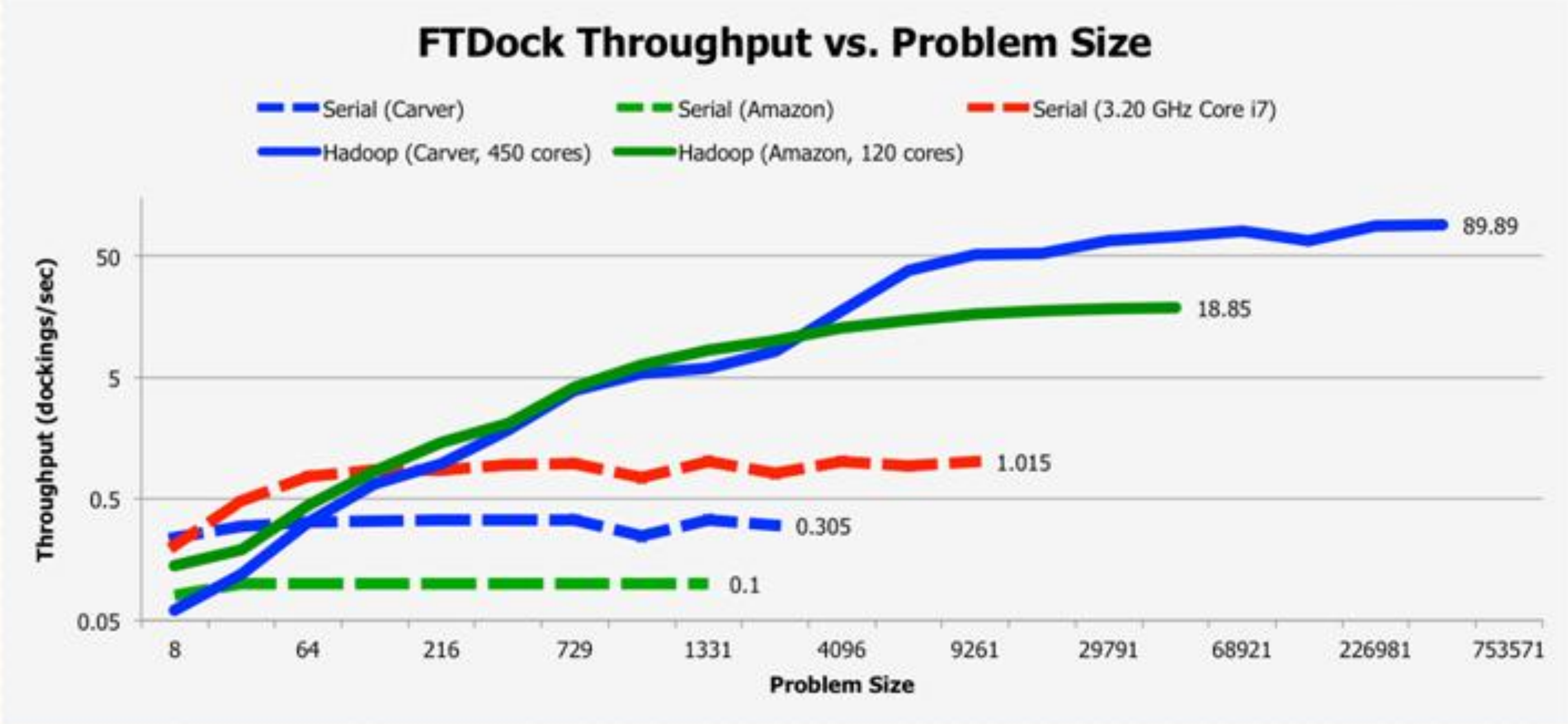
Incorporating new specializers

FTDock – Protein Docking

- Independent dockings in 3D search space
- Requires one-line change to application.
- Achieves **290x speedup** on 450 cores.

FTDock Specializer Core

```
class FtdockMRJob(AspMRJob):
  def mapper(self, coords, ignored):
    args = self.data['protein_data']
    score = ftdock(*coords, *args)
    yield 1, score
```



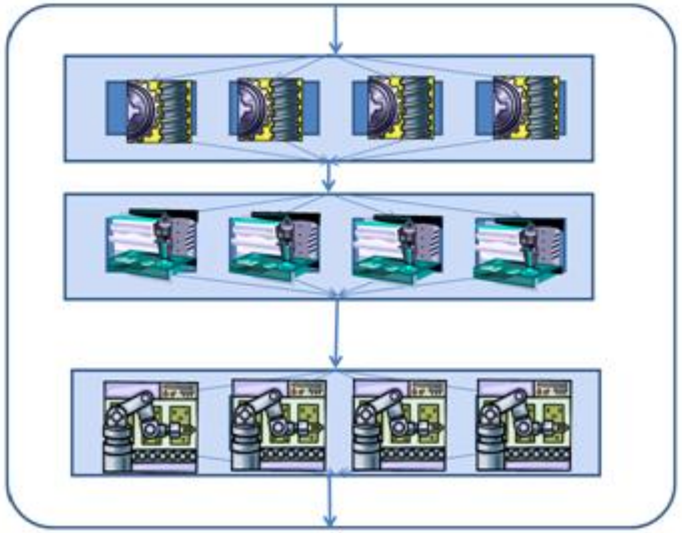
Source: M. Driscoll, E. Georgana, P. Koanantakool, 2012 ParLab winter Retreat.

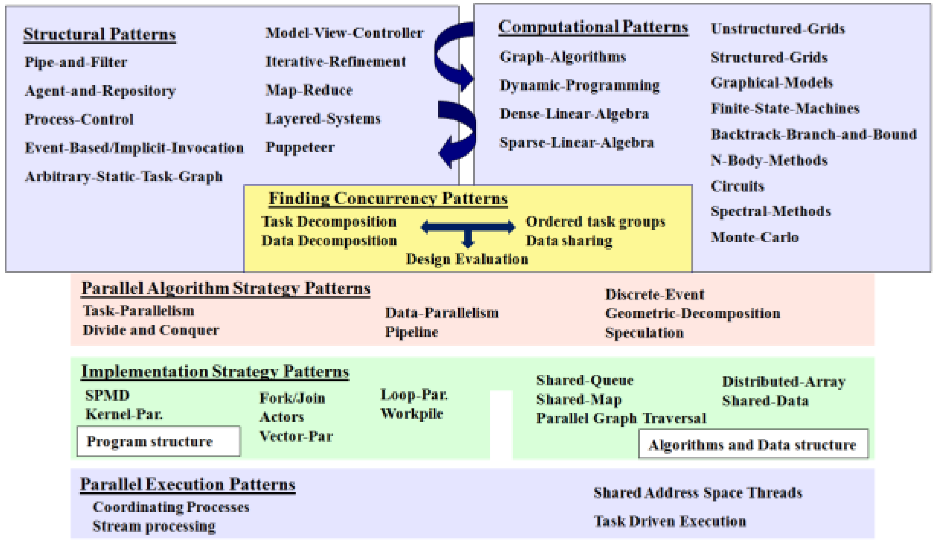
More Complicated Applications of SEJITS

- Complex interfaces to optimized libraries:
  - JIT'ed insertion of FFTW (accommodate APIs, build plans, clean up when done)
- Interface to auto-tuning:
  - Runtime auto-tuning to optimize library routines.
  - Cached so subsequent uses avoid auto-tuning overhead.
- Family of specializers to support other computational patterns:
  - Stencil
  - Graph algorithms
  - Graphical models
  - … and over time we'll fill in framework elements for all structural and computational patterns

Conclusion

- Understanding software architecture is how we will solve the many core programming challenge.
- An architecture is analogous to a factory … a structural arrangement of computational elements





- We define software architecture in terms of a pattern language called OPL.
  - Architectural patterns:
    - Structural patterns
    - Computational patterns
  - Parallel programming patterns (PLPP):
    - Algorithm strategy
    - Implementation strategy
    - Parallel execution Patterns

Microsoft