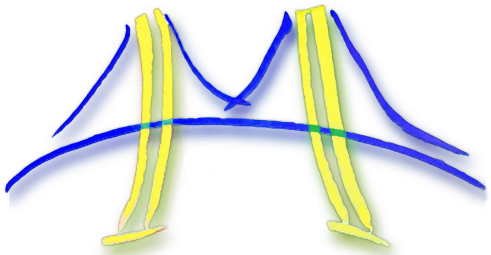


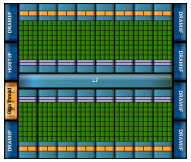
Architecting Parallel Software with Patterns

Kurt Keutzer, EECS, Berkeley
with thanks to Tim Mattson, Intel
and the PALLAS team:
Michael Anderson, Ekaterina Gonina, Patrick Li,
David Sheffield, Bor-Yiing Su, and Naryanan Sundaram,

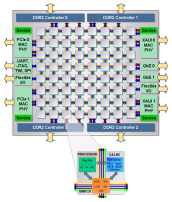
The Challenge of Parallelism

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

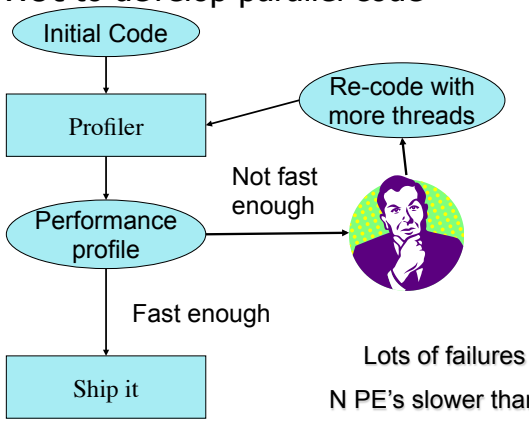
 2

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

3

Assumption #1:
How not to develop parallel code



4

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?

Edward Lee, "The Problem with Threads"

Threads, locks, semaphores, data races

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 techology of 25 software products, I came to the conclusion that:

- **Software architecture >> software environment**
- **Software environment >> programming language**

Software architecture: where we begin

Grady Booch
Object-Oriented Guru

Can be built by one person
Requires
- **Minimal modeling**
- **Simple process**
- **Simple tools**

10

The progress of Object Oriented Programming

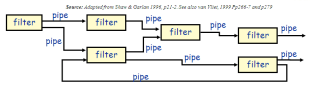
Built most efficiently and timely by a team
Requires
- **Modeling**
- **Well-defined process**
- **Power tools**

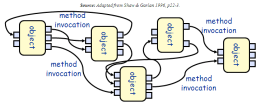
Grady Booch
OO Guru

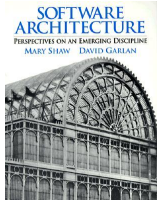
11

Architectural Styles (Garland and Shaw, 1996)

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







Civil Architectural Styles

Problem: Build a residential home:

Architectural styles:

- **Ranch house**
- **Victorian**
- **Colonial**
- **A-frame**
- **Bungalow**
- **Cape cod**

 13

Goal – Future sw architecture

Grady Booch
OO Guru

Progress
- **Advances in materials**
- **Advances in analysis**

Scale
- **5 times the span of the Pantheon**
- **3 times the height of Cheops**

14

How does Modularity Help?

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**
- **Modularity helps us to identify useful invariants and 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.**

Object-Oriented Programming

Focused on:

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

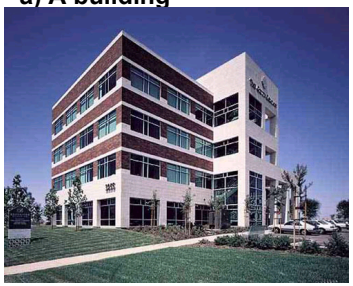
Neglected:

- **Application concurrency**
- **Computational details**
- **Parallel implementations**

17

Pop Quiz: Is a software program more like?

a) A building



b) A factory

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

Another piece of the puzzle from the HPC community

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

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

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

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

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

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

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

19

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.

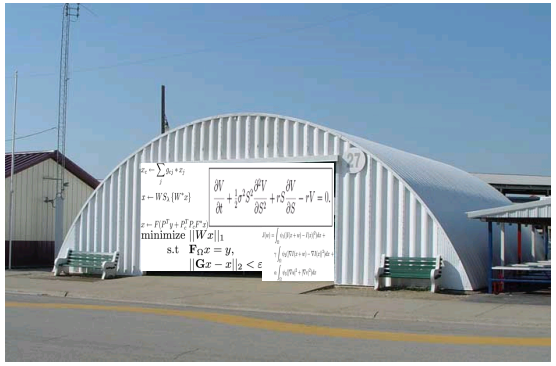
Remainder of this talk will consider one of them (structured grids) in detail.

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												

Unfortunately … HPC approach to software architecture architecture

Technically this is known as a *monolithic* architecture



23

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**

24

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

25

Alexander's Pattern Language

Christopher Alexander's approach to (civil) architecture:

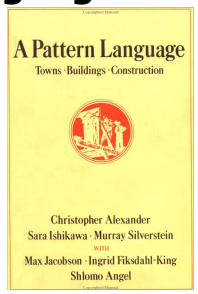
- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." *Page x, A Pattern Language,* Christopher 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!!!



26

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 new ideas – approaches you may not have considered
  - Gives a set of finiteness – 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

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	Identify the Key Computations
•Pipe-and-Filter	• Graph Algorithms
•Agent-and-Repository	• Dynamic programming
•Event-based	• Dense/Spare Linear Algebra
•Process Control	• (Un)Structured Grids
•Layered Systems	• Graphical Models
• Model-view controller	• Finite State Machines
•Iterator	• Backtrack Branch-and-Bound
•MapReduce	• N-Body Methods
•Arbitrary Task Graphs	• Circuits
•Puppeteer	• Spectral Methods

28

Architecting Parallel Software

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

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

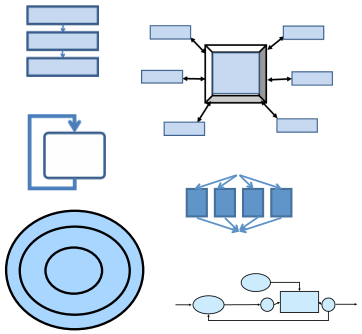
Identify the Software Structure ⇄ Identify the Key Computations

29

Identify the SW Structure

Structural Patterns

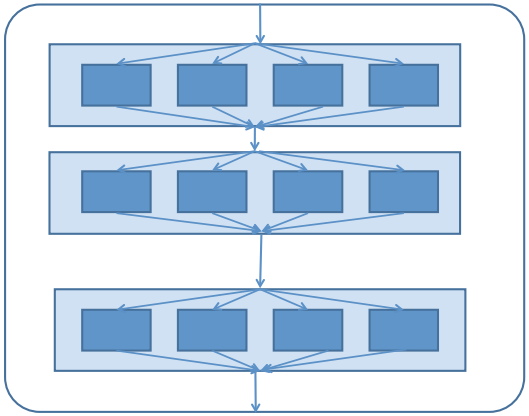
- Pipe-and-Filter
- Agent-and-Repository
- Event-based coordination
- Iterator
- MapReduce
- Process Control
- Layered Systems



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

30

Analogy: Layout of Factory Plant



31

Identify key computations ….

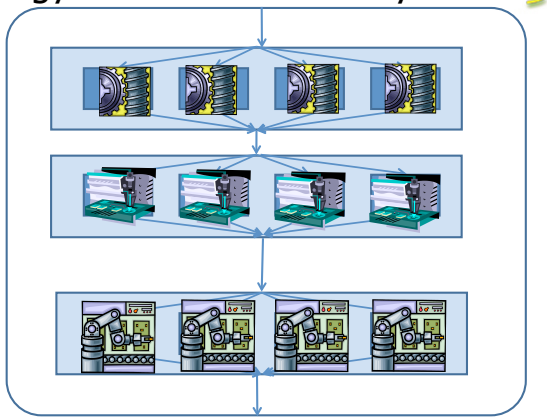
Apps / Dwarves	Embed	SPEC	DB	Games	ML	HPC	CAD	Health	Image	Speech	Music	Browser
Graph Algorithms												
Graphical Models												
Backtrack / B&B												
Finite State Mach.												
Circuits												
Dynamic Prog.												
Unstructured Grid												
Structured Grid											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

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

Analogy: Machinery of the Factory

33

Analogy: Architected Factory



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

34

Architecting Parallel Software

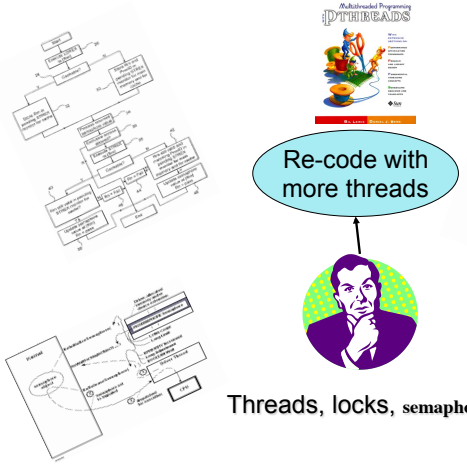
Structural Patterns

- **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**

Hint: What is this person thinking of?



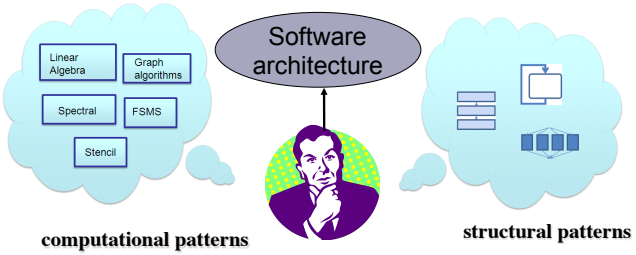
Edward Lee, "The Problem with Threads"

Threads, locks, **semaphores, data races**

36

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



37

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

38

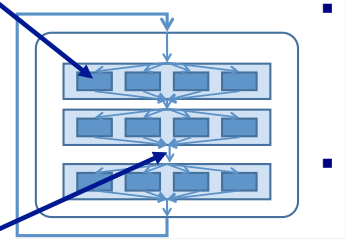
Inventory of Structural Patterns

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

39

Elements of a structural pattern

- Components are where the computation happens



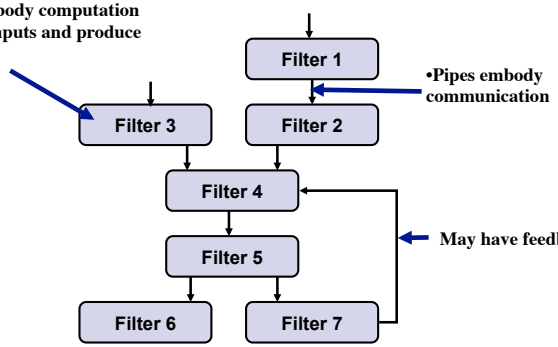
- A configuration is a graph of components (vertices) and connectors (edges)
- A structural patterns may be described as a familiy of graphs.

Connectors are where the communication happens

40

Pattern 1: Pipe and Filter

•Filters embody computation
•Only see inputs and produce outputs

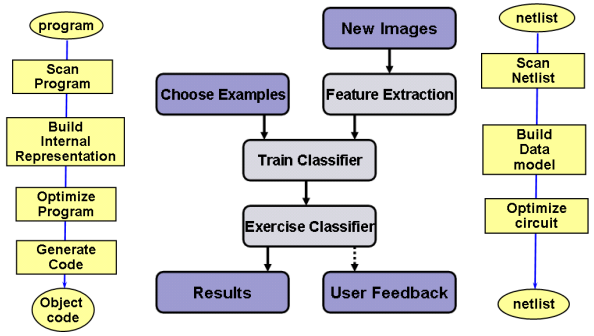


Examples?

41

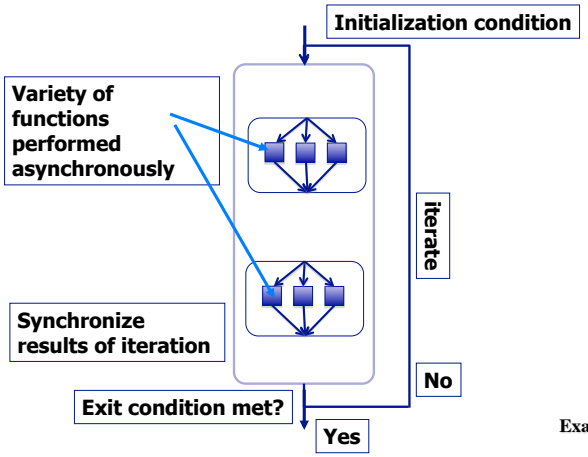
Examples of pipe and filter

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



42

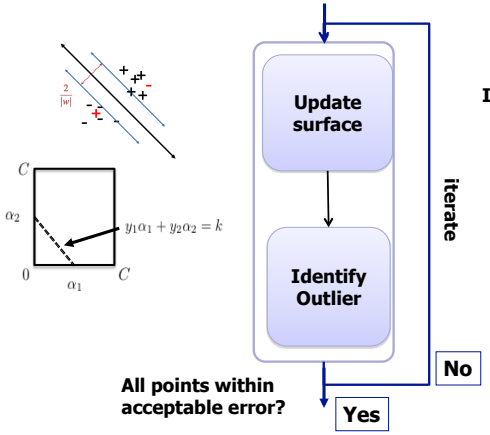
Pattern 2: Iterator Pattern



Examples?

43

Example of Iterator Pattern: Training a Classifier: SVM Training



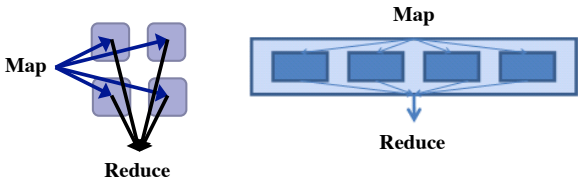
Iterator Structural Pattern

44

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)**

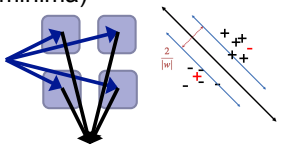


Examples?

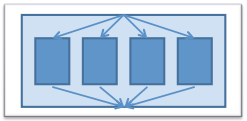
45

Examples of Map Reduce

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



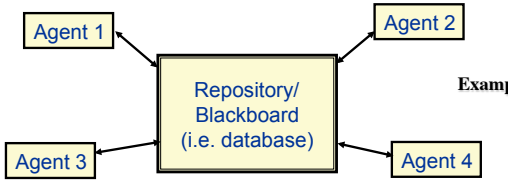
Support-vector machines (ML)
• Map to evaluate distance from the frontier
• Reduce to find the greatest outlier from the frontier



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

46

Pattern 4: Agent and Repository



Examples?

Agent and repository : Blackboard structural pattern

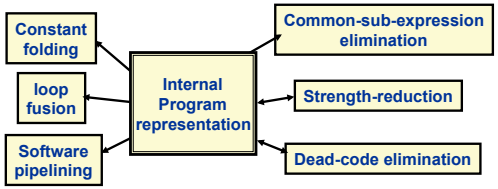
Agents cooperate on a shared medium to produce a result

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)**

47

Example: Compiler Optimization

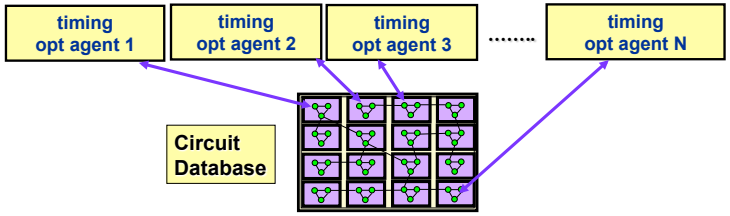


Optimization of a software program

- **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**

48

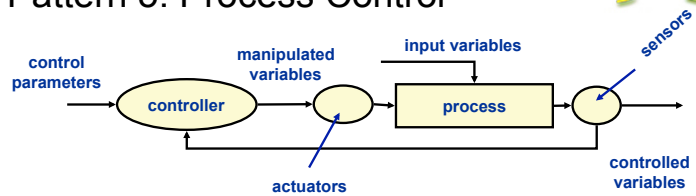
Example: Logic Optimization



- 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

49

Pattern 5: Process Control



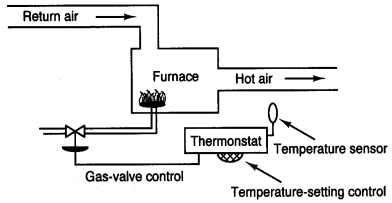
Source: Adapted from Shaw & Garlan 1996, p27-31.

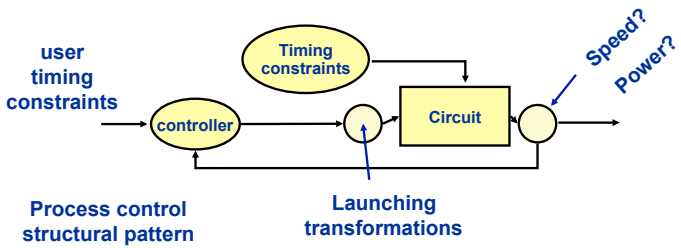
- Process control:
  - **Process: underlying phenomena to be controlled/computed**
  - **Actuator: task(s) affecting the process**
  - **Sensor: task(s) which analyze the state of the process**
  - **Controller: task which determines what actuators should be effected**

Examples?

50

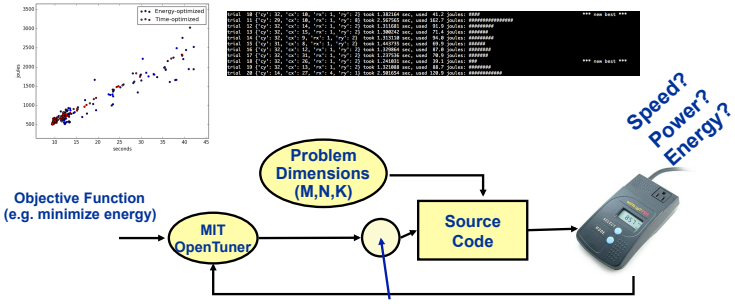
Examples of Process Control





51

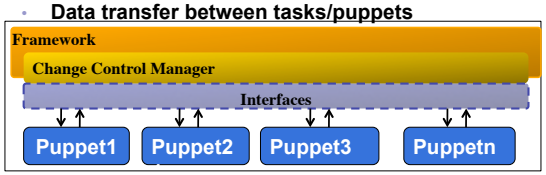
Examples of Process Control: Matrix Multiply Autotuner (Michael Driscoll)



52

Pattern 9: Puppeteer

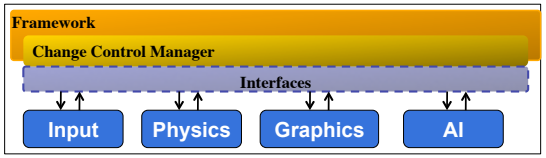
- Need an efficient way to manage and control the interaction of multiple simulators/computational agents
- Puppeteer Pattern – guides the interaction between the tasks/ puppets to guarantee correctness of the overall task
- Puppeteer: 1) schedules puppets 2) manages exchange of data between puppets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



Examples?

53/17

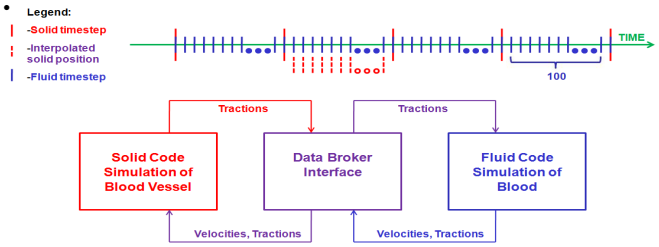
Video Game



54/17

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



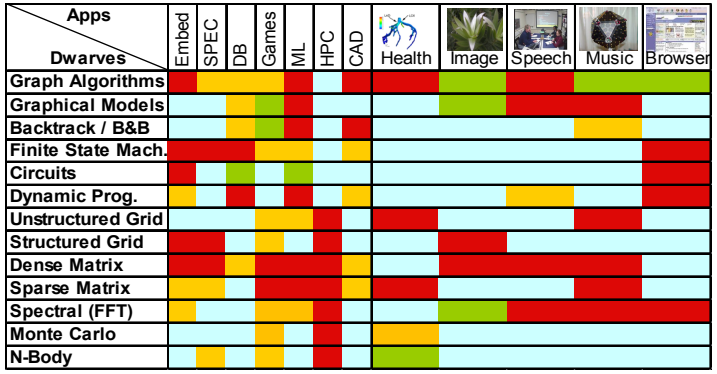
55

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

56

You explore these every class

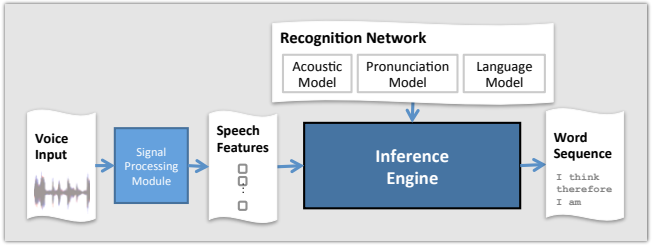


Outline

- 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

58

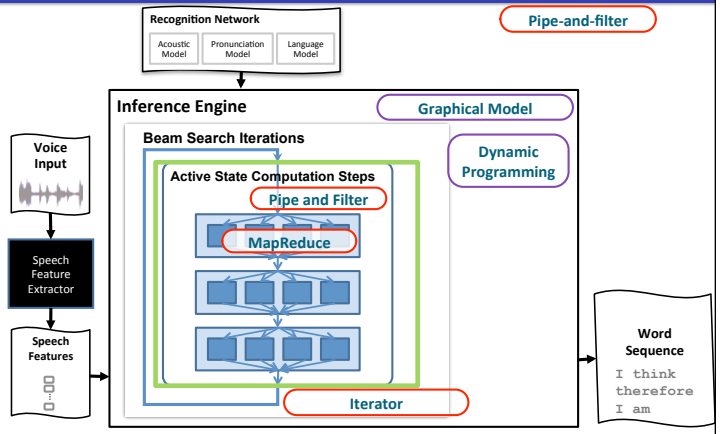
Large Vocabulary Continuous Speech Recognition



- 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

59/69

LVCSR Software Architecture



60/69

Key computation: HMM Inference Algorithm

An instance of: Graphical Models

Implemented with: Dynamic Programming

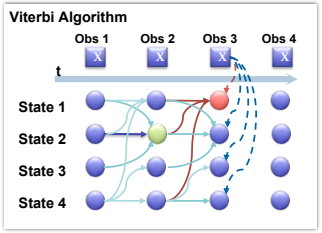
- Finds the most-likely sequence of states that produced the observation

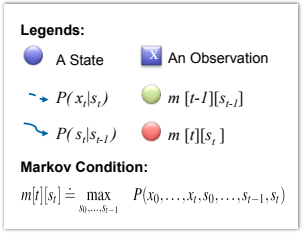
$$m[t][s_t] = \max_{s_{t-1}} m[t-1][s_{t-1}] \cdot P(s_t|s_{t-1}) \cdot P(x_t|s_t)$$

GMM

Frontier

Rec Network Transition Probability



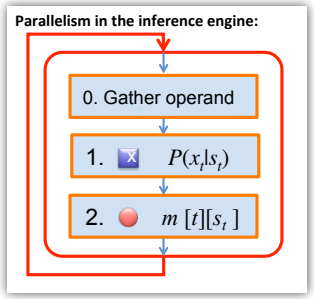


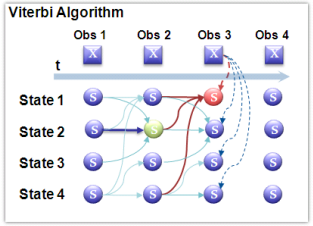
J. Chong, Y. Yi, A. Faria, N.R. Satish and K. Keutzer, "**Data-Parallel Large Vocabulary Continuous Speech Recognition on Graphics Processors**", Emerging Applications and Manycore Arch. 2008, pp. 23-35, June 2008

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

0. Gather operand

max

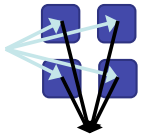
63/69

Each Filter is Map Reduce
1. observation probability computation

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

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

1. $P(x_t|s_t)$



max

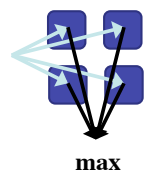
64/69

Each Filter is Map Reduce
2. graph traversal computation

PARALLAS

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

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

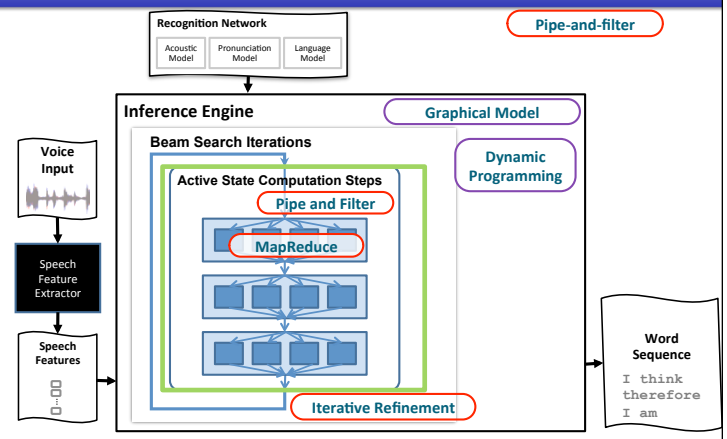


2. $m[t][s_t]$

65/69

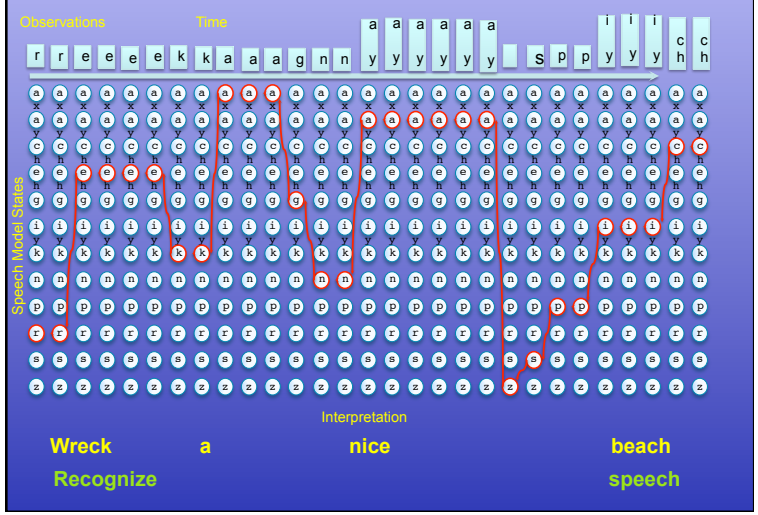
LVCSR Software Architecture

PARALLAS



66/69

HMM computed with Dynamic Programming



This Approach Works

PARALLAS

Application	Speedups	
MRI	100x	IEEE TMI 2012
SVM-train (>2900 Downloads)	20x	ICML 2008
SVM-classify	109x	
Contour (>2900 Downloads)	130x	ICCV 2009
Object Recognition	80x	WACV 2011
Poselet	20x	
Optical Flow	32x	ECCV 2010
Speech	11x	Interspeech 2010, 2011
Value-at-risk	60x	Wiley 2011
Option Pricing	25x	

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

68/69

Outline

PARLAS

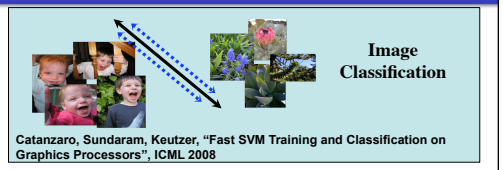
- What doesn't work
- Pieces of the problem … and solution
- General approach to architecting parallel sw
- Detail on Structural Patterns
- Detail on Computational Patterns
- High-level examples of architecting applications
- ⇨ Summary

69/69

Recap: Architecting Parallel Software

PARLAS

1. Start with a compelling, performance sensitive application.

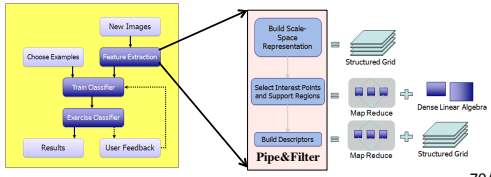


2. Define the overall structure

Identify the Software Structure ⇄ **Identify the Key Computations**

3. Define computations inside structural elements

4. Compose Structural and computational patterns to yield software architecture

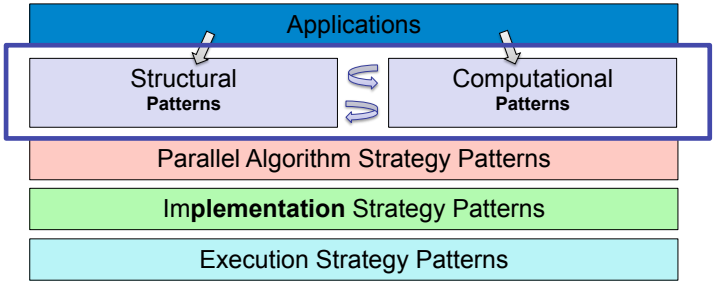


"Image Feature Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC '09

70/69

Our Pattern Language

PARLAS



71/69

Computational Patterns Make me Feel Smart

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												

- 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

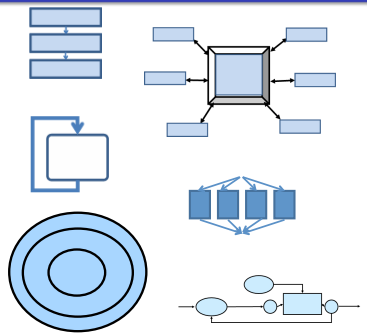
72/69

Structural Patterns Make me Feel Organized

PALLAS

Structural Patterns

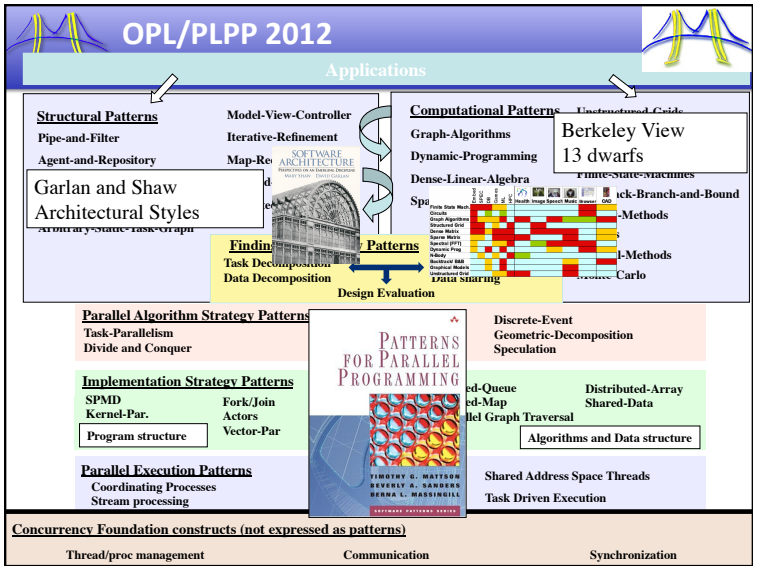
- Pipe-and-Filter
- Agent-and-Repository
- Event-based
- Layered Systems
- Model-view-controller
- Arbitrary Task Graphs
- Puppeteer
- Iterator/BSP
- MapReduce



- The modularity provided by structural patterns make me feel organized.
- **Even the most complex application can be broken down into manageable modules**

73/69

OPL/PLPP 2012



Summary

PALLAS

- 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 five – five more will be all you 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:

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

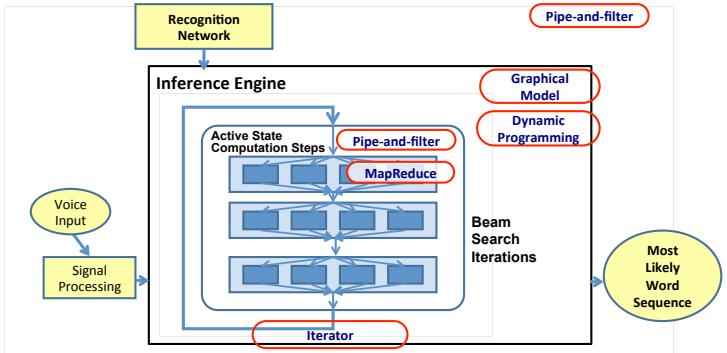
PALLAS: http://parlab.eecs.berkeley.edu/research/pallas

75/69

More examples

76

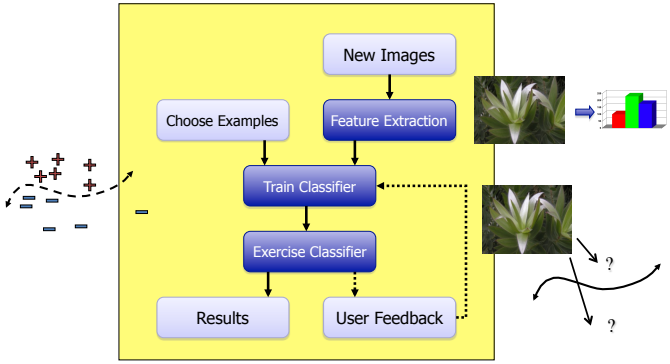
Architecting Speech Recognition



Large Vocabulary Continuous Speech Recognition Poster: Chong, Yi
Work also to appear at Emerging Applications for Manycore Architecture

77

CBIR Application Framework



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

78

Feature Extraction

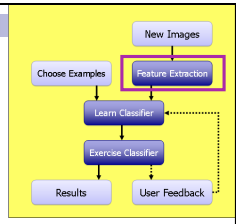
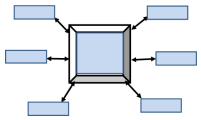


Image histograms are common to many feature extraction procedures, and are an important feature in their own right

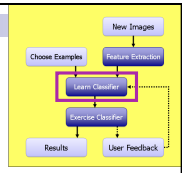


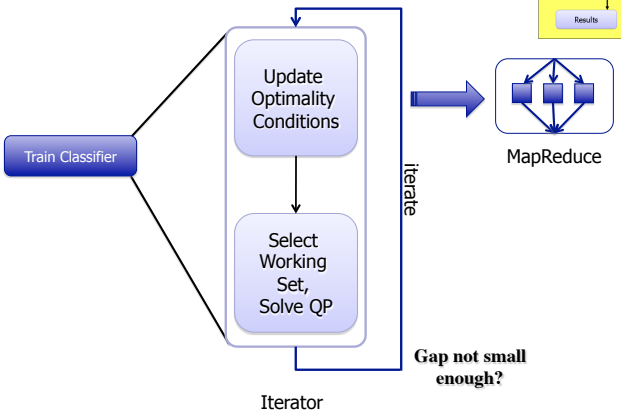
- Agent and Repository: Each agent computes a local transform of the image, plus a local histogram.
- Results are combined in the repository, which contains the global histogram

- The data dependent access patterns found when constructing histograms make them a natural fit for the agent and repository pattern

79

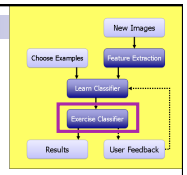
Train Classifier: SVM Training

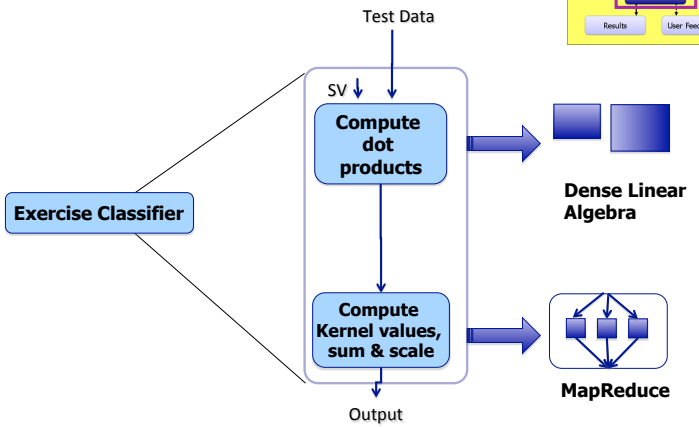




80

Exercise Classifier : SVM Classification





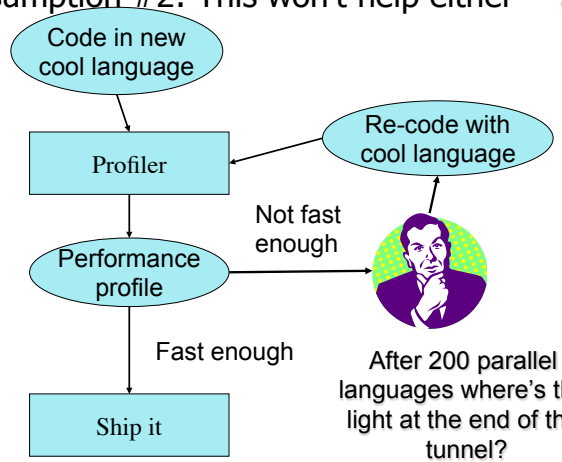
81

Key Elements of Kurt's SW Education

- AT&T Bell Laboratories: CAD researcher and programmer
  - Algorithms: D. Johnson, R. Tarjan
  - Programming Pearls: S C Johnson, K. Thompson, (Jon Bentley)
  - Developed useful software tools:
    - Plaid: programmable logic aid: used for developing 100's of FPGA-based HW systems
    - CONES/DAGON: used for designing >30 application-specific integrated circuits
- Synopsys: researcher → CTO (25 products, ~15 million lines of code, $750M annual revenue, top 20 SW companies)
  - Super programming: J-C Madre, Richard Rudell, Steve Tjiang
  - Software architecture: Randy Allen, Albert Wang
  - High-level Invariants: Randy Allen, Albert Wang
- Berkeley: teaching software engineering and Par Lab
  - Took the time to reflect on what I had learned:
  - Architectural styles: Garlan and Shaw
    - Design patterns: Gamma et al (aka Gang of Four), Mattson's PLPP
    - A Pattern Language: Alexander, Mattson
    - Dwarfs: Par Lab Team

82

Assumption #2: This won't help either



83

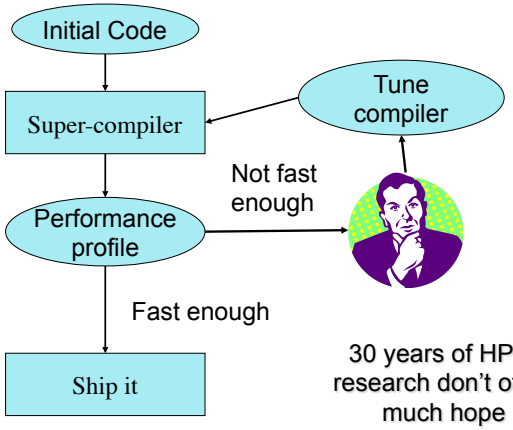
83

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
AMDC	DCE++	JAVAR	Modula-2*	pC++	smalltalk
AppLeS	DDD	JADE	Multipol	PCN	SML
Amoeba	DICE.	Java RMI	MPI	PCP:	SONiC
ARTS	DIPC	javaPG	MPC++	PH	Split-C.
Athapascan-0b	DOLIB	JavaSpace	Munin	PEACE	SR
Aurora	DOME	JIDL	Nano-Threads	PCU	Sthreads
Automap	DOSMOS.	Joyce	NESL	PET	Strand.
bb_threads	DRL	Khoros	NetClasses++	PETSc	SUIF.
Blaze	DSM-Threads	Karma	Nexus	PENNY	Synergy
BSP	Ease .	KOAN/Fortran-S	Nimrod	Phosphorus	Telegrphos
BlockComm	ECO	LAM	NOW	POET.	SuperPascal
C*.	Eiffel	Lilac	Objective Linda	Polaris	TCGMSG.
"C* in C	Eilean	Linda	Occam	POOMA	Threads.h++.
C**	Emerald	JADA	Omega	POOL-T	TreadMarks
CarlOS	EPL	WWWinda	OpenMP	PRESTO	TRAPPER
Cashmere	Excalibur	ISETL-Linda	Orca	P-RIO	uC++
C4	Express	ParLin	OOF90	Prospero	UNITY
CC++	Falcon	Eilean	P++	Proteus	UC
Chu	Filaments	P4-Linda	P3L	QPC++	V
Charlotte	FM	Glenda	p4-Linda	PVM	ViC*
Charm	FLASH	POSYBL	Pablo	PSI	Visifold V-NUS
Charm++	The FORCE	Objective-Linda	PADE	PSDM	VPE
Cid	Fork	LiPS	PADRE	Quake	Win32 threads
Cilk	Fortran-M	Locust	Panda	Quark	WinPar
CM-Fortran	FX	Lparx	Papers	Quick Threads	WWWinda
Converse	GA	Lucid	AFAPI.	Sage++	XENOOPS
Code	GAMMA	Maisie	Para++	SCANDAL	XPC
COOL	Glenda	Manifold	Paradigm	SAM	Zounds
					ZPL

84

Assumption #3: Nor this

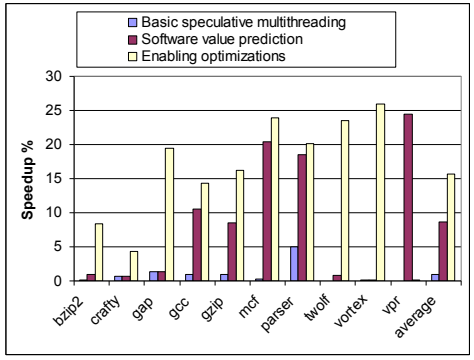


30 years of HPC research don't offer much hope

85

85

Automatic parallelization?



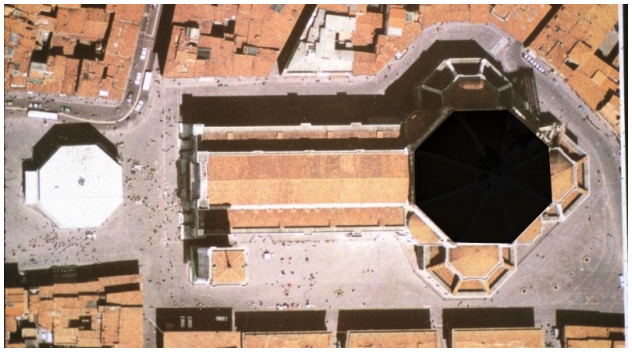
- Aggressive techniques such as speculative multithreading help, but they are not enough.
- Ave SPECint speedup of 8% … will climb to ave. of 15% once their system is fully enabled.
- There are no indications auto par. will radically improve any time soon.
- Hence, I do not believe Auto-par will solve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

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

86

Reinvention of design?

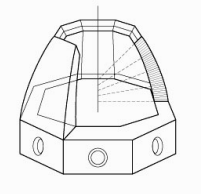
- In 1418 the **Santa Maria del Fiore** stood without a dome.
- Brunelleschi won the competition to finish the dome.
- Construction of the dome without the support of flying buttresses seemed unthinkable.

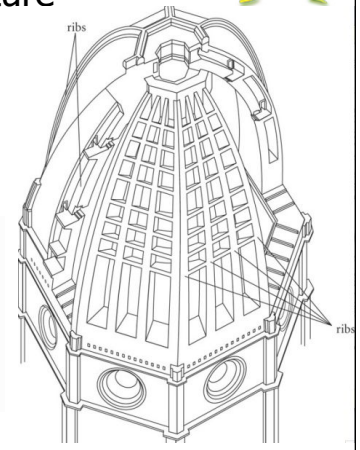


87

Innovation in architecture

- After studying earlier Roman and Greek architecture, Brunelleschi drew on diverse architectural styles to arrive at a dome design that could stand independently



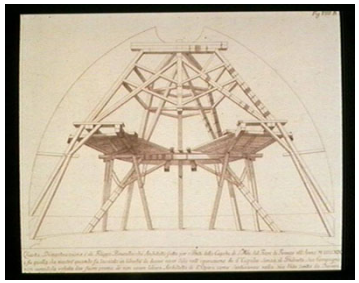


http://www.templejc.edu/dept/Art/ASmith/ARTS1304/Joe1/ZoomSlide0010.html

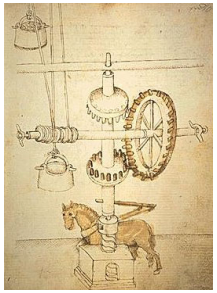
88

Innovation in tools

- His construction of the dome design required the development of new tools for construction, as well as an early (the first?) use of architectural drawings (now lost).



Scaffolding for cupola



Mechanism for raising materials

http://www.artist-biography.info/gallery/filippo_brunelleschi/67/

89

Innovation in use of building materials

- His construction of the dome design also required innovative use of building materials.

Herringbone pattern bricks

http://www.buildingstonemagazine.com/winter-06/art/dome8.jpg

90

Resulting Dome

Completed dome

http://www.duomofirenze.it/storia/cupola_eng.htm

91

The point?

- Challenges to design and build the dome of Santa Maria del Fiore showed underlying weaknesses of architectural understanding, tools, and use of materials
- By analogy, parallelizing code should not have thrown us for such a loop. Our difficulties in facing the challenge of developing parallel software are a symptom of underlying weakness is in our abilities to:
  - Architect software
  - Develop robust tools and frameworks
  - Re-use implementation approaches
- Time for a serious rethink of all of software design

92