PARLab Parallel Boot Camp

Architecting Parallel Software with Patterns

Kurt Keutzer
Electrical Engineering and Computer Sciences
University of California, Berkeley
Assumption #1: How not to develop parallel code

Initial Code

Profiler

Performance profile

Re-code with more threads

Not fast enough

Fast enough

Lots of failures

N PE’s slower than 1

Ship it
# Steiner Tree Construction Time By Routing Each Net in Parallel

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Serial</th>
<th>2 Threads</th>
<th>3 Threads</th>
<th>4 Threads</th>
<th>5 Threads</th>
<th>6 Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>adaptec1</td>
<td>1.68</td>
<td>1.68</td>
<td>1.70</td>
<td>1.69</td>
<td>1.69</td>
<td>1.69</td>
</tr>
<tr>
<td>newblue1</td>
<td>1.80</td>
<td>1.80</td>
<td>1.81</td>
<td>1.81</td>
<td>1.81</td>
<td>1.82</td>
</tr>
<tr>
<td>newblue2</td>
<td>2.60</td>
<td>2.60</td>
<td>2.62</td>
<td>2.62</td>
<td>2.62</td>
<td>2.61</td>
</tr>
<tr>
<td>adaptec2</td>
<td>1.87</td>
<td>1.86</td>
<td>1.87</td>
<td>1.88</td>
<td>1.88</td>
<td>1.88</td>
</tr>
<tr>
<td>adaptec3</td>
<td>3.32</td>
<td>3.33</td>
<td>3.34</td>
<td>3.34</td>
<td>3.34</td>
<td>3.34</td>
</tr>
<tr>
<td>adaptec4</td>
<td>3.20</td>
<td>3.20</td>
<td>3.21</td>
<td>3.21</td>
<td>3.21</td>
<td>3.21</td>
</tr>
<tr>
<td>adaptec5</td>
<td>4.91</td>
<td>4.90</td>
<td>4.92</td>
<td>4.92</td>
<td>4.92</td>
<td>4.92</td>
</tr>
<tr>
<td>newblue3</td>
<td>2.54</td>
<td>2.55</td>
<td>2.55</td>
<td>2.55</td>
<td>2.55</td>
<td>2.55</td>
</tr>
<tr>
<td><strong>average</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.0011</strong></td>
<td><strong>1.0044</strong></td>
<td><strong>1.0049</strong></td>
<td><strong>1.0046</strong></td>
<td><strong>1.0046</strong></td>
</tr>
</tbody>
</table>
Assumption #2: This won’t help either

Code in new cool language

Profiler

Performance profile

Fast enough

Ship it

Re-code with cool language

Not fast enough

After 200 parallel languages where’s the light at the end of the tunnel?
### Parallel Programming environments in the 90's

<table>
<thead>
<tr>
<th>ABCPL</th>
<th>CORRELATE</th>
<th>GUARD</th>
<th>GLU</th>
<th>Mental</th>
<th>ParallelO</th>
<th>pC++</th>
<th>SCHEDULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE</td>
<td>CPS</td>
<td>HAXL</td>
<td>Meta Chase</td>
<td>Millipede</td>
<td>ParC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACT++</td>
<td>CRIL</td>
<td>HPC</td>
<td>HPC</td>
<td>Metasite</td>
<td>Millipede</td>
<td>ParC++</td>
<td></td>
</tr>
<tr>
<td>Activemessages</td>
<td>Cthreads</td>
<td>HPC++</td>
<td>HPC</td>
<td>Metasite</td>
<td>Millipede</td>
<td>ParC++</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>CUMULUS</td>
<td>JAVA</td>
<td>Java RMI</td>
<td>HPCC++</td>
<td>ParC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addon</td>
<td>DAISY</td>
<td>JavaPG</td>
<td>Java</td>
<td>HPCC++</td>
<td>ParC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDAP</td>
<td>DAGGER</td>
<td>JavaPG</td>
<td>Java</td>
<td>HPCC++</td>
<td>ParC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AFAP</td>
<td>DAPPLE</td>
<td>JavaPG</td>
<td>Java</td>
<td>HPCC++</td>
<td>ParC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLWAN</td>
<td>Data Parallel C</td>
<td>IMPACT</td>
<td>MOSIX</td>
<td></td>
<td>ParC++</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM</td>
<td>DC++</td>
<td>IS86</td>
<td>Module-P</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMDC</td>
<td>DCE++</td>
<td>JAVA</td>
<td>Module-2</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AppL++</td>
<td>DDD</td>
<td>JAVA</td>
<td>Multipol</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amelab</td>
<td>DICE</td>
<td>JAVA</td>
<td>MPI</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTS</td>
<td>DOLIB</td>
<td>JAVA</td>
<td>MPI</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Athanasos-6b</td>
<td>DOME</td>
<td>JAVA</td>
<td>MPIC++</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aurora</td>
<td>DOME</td>
<td>JAVA</td>
<td>MPIC++</td>
<td>pC++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aurum</td>
<td>DOSMOS</td>
<td>JAVA</td>
<td>Nano-Threads</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automap</td>
<td>DRL</td>
<td>JAVA</td>
<td>Nano-Threads</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bcc_threads</td>
<td>DRM</td>
<td>JAVA</td>
<td>Nano-Threads</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blaze</td>
<td>DSM-Threads</td>
<td>JAVA</td>
<td>Nexus</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSP</td>
<td>Base</td>
<td>KOAN/Forest-II</td>
<td>Nexus</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BlockComm</td>
<td>ECO</td>
<td>LAND</td>
<td>OpenMP</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>Effel</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;C++ in C&quot;</td>
<td>Effel</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C++</td>
<td>Eilene</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CarIS</td>
<td>EPL</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cashmere</td>
<td>Enclave</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C#++</td>
<td>Envelope</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC++</td>
<td>Envelope</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chen</td>
<td>Envelope</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlotte</td>
<td>FUTE</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charm++</td>
<td>FLASH</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charm++</td>
<td>The FORCE</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilok</td>
<td>Fork</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk</td>
<td>Fortran-M</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cilk-Fortran</td>
<td>SX</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converse</td>
<td>GA</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>GADIA</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COOL</td>
<td>Glenda</td>
<td>Linux</td>
<td>Objectiv</td>
<td>pC++</td>
<td>RESI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Assumption #3: Nor this

Initial Code

Super-compiler

Performance profile

Tune compiler

Not fast enough

Fast enough

Ship it

30 years of HPC research don’t offer much hope
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.
Outline

- Intro to Kurt
  - General approach to applying the pattern language
  - Detail on Structural Patterns
  - High-level examples of composing patterns
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
What I learned (the hard way)

- Software must be architected to achieve productivity, efficiency, and correctness
- SW architecture >> programming environments
  - >> programming languages
  - >> compilers and debuggers
  - (>>hardware architecture)
- Discussions with superprogrammers taught me:
  - Give me the right program structure/architecture I can use any programming language
  - Give me the wrong architecture and I'll never get there
- What I've learned when I had to teach this stuff at Berkeley:
  - Key to architecture (software or otherwise) is design patterns and a pattern language
  - Resulting software design then uses a hierarchy of software frameworks for implementation
    - Application frameworks for application (e.g. CAD) developers
    - Programming frameworks for those who build the application frameworks
Outline

- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns
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!!!
Alexander’s Pattern Language (95-103)

• Layout the overall arrangement of a group of buildings: the height and number of these buildings, the entrances to the site, main parking areas, and lines of movement through the complex.

• 95. Building Complex
• 96. Number of Stories
• 97. Shielded Parking
• 98. Circulation Realms
• 99. Main Building
• 100. Pedestrian Street
• 101. Building Thoroughfare
• 102. Family of Entrances
• 103. Small Parking Lots
Family of Entrances (102)

- May be part of Circulation Realms (98).
- Conflict:
- When a person arrives in a complex of offices or services or workshops, or in a group of related houses, there is a good chance he will experience confusion unless the whole collection is laid out before him, so that he can see the entrance place where he is going.

Resolution:
Lay out the entrances to form a family. This means:
- 1) They form a group, are visible together, and each is visible from all the others.
- 2) They are all broadly similar, for instance all porches, or all gates in a wall, or all marked by a similar kind of doorway.
- May contain Main Entrance (110), Entrance Transition (112), Entrance Room (130), Reception Welcomes You (149).
Family of Entrances

http://www.intbau.org/Images/Steele/Badran5a.jpg
Elements of a Pattern - 1

• Name
  - It must have a meaningful name useful to remember the pattern and when it is used.

• Problem
  - A statement of the problem ... a one-line preamble and the problem stated as a question.

• Context
  - The conditions under which the problem occurs. Defines when the pattern is applicable and the configuration of the system before the pattern is applied.

• Forces
  - A description of the relevant forces and constraints and how they interact/conflict with one another and with goals we wish to achieve. Defines the tension that characterizes a problem.
Elements of a Pattern - 2

• Solution
  - Instructions used to solve the problem. When done right, it resolves the tension defined in the forces section; flowing from the context and forces. We also define the new context for the system following application of the pattern.

• Invariant
  - What must be invariantly true for this pattern to work. May be stated in the form of Precondition, Invariant, Post-condition

• Examples
  - Examples to help the reader understand the pattern.

• Known Uses and frameworks
  - Cases where the pattern was used; preferably with literature references.

• Related Patterns
  - How does this pattern fit-in or work-with the other patterns in the pattern language.

*PLPP is the pattern language from Mattson, Sanders and Massingill
Computational Patterns

( Red Hot ➔ Blue Cool )
PLPP is the first attempt to develop a complete *pattern language* for parallel software development.

PLPP is a great model for a pattern language for parallel software.

PLPP mined scientific applications that utilize a monolithic application style.

PLPP doesn’t help us much with horizontal composition.

Much more useful to us than: *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma, Helm, Johnson & Vlissides, Addison-Wesley, 1995.
Structural programming patterns

- In order to create more complex software it is necessary to compose programming patterns
- For this purpose, it has been useful to induct a set of patterns known as “architectural styles”

Examples:
- pipe and filter
- event based/event driven
- layered
- Agent and repository/blackboard
- process control
- Model-view-controller
Put it all together
Our Pattern Language 2.0: Keutzer and Mattson

Applications

Choose your high level structure - what is the structure of my application? Guided expansion
- Pipe-and-filter
- Agent and Repository
- Process Control
- Event based, implicit invocation

Choose you high level architecture? Guided decomposition
- Task Decomposition ↔ Data Decomposition
- Group Tasks
- Order groups
- data sharing
- data access

Model-view controller
- Iteration
- Map reduce
- Layered systems
- Arbitrary Static Task
- Graph

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

Refine the structure - what concurrent approach do I use? Guided re-organization
- Event Based
- Divide and Conquer
- Data Parallelism
- Geometric Decomposition
- Pipeline
- Discrete Event
- Task Parallelism
- Graph algorithms
- Digital Circuits

Utilize Supporting Structures - how do I implement my concurrency? Guided mapping
- Fork/Join
- Distributed Array
- Shared Data
- Shared Queue
- Shared Hash Table
- SPMD
- Master/worker
- Loop Parallelism
- BSP

Implementation methods - what are the building blocks of parallel programming? Guided implementation
- Thread Creation/destruction
- Message passing
- Speculation
- Barriers
- Transactional memory
- Semaphores
- Mutex
Our Pattern Language 2.0

Applications

Choose your high level structure - what is the structure of my application? Guided

Graph Algorithms
Dynamic Programming
Dense Linear Algebra
Sparse Linear Algebra
Unstructured Grids
Structured Grids

Choose you high level architecture? Guided decomposition

Task Decomposition → Data Decomposition
Group Tasks Order groups data sharing data access

Identify the key computational patterns - what are my key computations?

Garlan and Shaw
Architectural Styles

Agent and Repository
Process Control
Event based, implicit invocation

Berkeley View

13 dwarfs

Backtrack Branch and Bound
N-Body methods
Circuits
Spectral Methods

Refine the structure - what concurrent approach do I use? Guided re-organization

Event Based
Data Parallelism
Geometric Decomposition

Task Parallelism
Graph algorithms

Utilize Supporting Structures - how do I implement my concurrency? Guided mapping

Fork/Join
Master/worker
Distributed Array

CSP
Shared Queue
SPMD

Loop Parallelism
Barriers
BSP

Implementation methods - what are the building blocks of parallel programming? Guided implementation

Thread Creation/destruction
Message passing
Collective communication

Process Creation/destruction
Transactional memory
Mutex
Architecting Parallel Software

Decompose Tasks/Data
Order tasks Identify Data Sharing and Access

Identify the Software Structure

Pipe-and-Filter
Agent-and-Repository
Event-based
Bulk Synchronous
MapReduce
Layered Systems
Arbitrary Task Graphs

Identify the Key Computations

Graph Algorithms
Dynamic programming
Dense/Spare Linear Algebra
(Un)Structured Grids
Graphical Models
Finite State Machines
Backtrack Branch-and-Bound
N-Body Methods
Circuits
Spectral Methods
Pop Quiz: Software is More Like ...

a) A building  
b) A factory
These define the structure of our software but they do not describe what is computed.
Analogy: Layout of Factory Plant
Identify Key Computations

- Computational patterns describe the key computations but not how they are implemented
Analogy: Machinery of the Factory
Architecting the Whole Application

- **SW Architecture of Large-Vocabulary Continuous Speech Recognition**

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

Analogous to the design of an entire manufacturing plant
Outline

- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns
Inventory of Structural Patterns

- pipe and filter
- iterator
- MapReduce
- blackboard/agent and repository
- process control
- layered
- event-based coordination
- puppeteer
- (call-and-return/arbitrary task graph)
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 family of graphs.

Connectors are where the communication happens.
Pattern 1: Pipe and Filter

- Filters embody computation
- Only see inputs and produce outputs

Pipes embody communication
May have feedback

Examples?
Almost every large software program has a pipe and filter structure at the highest level.
Pattern 2: Iterator Pattern

- Variety of functions performed asynchronously
- Initialization condition
- Synchronize results of iteration
- Exit condition met?
  - Yes
  - No

Examples?
Example of Iterator Pattern: Training a Classifier: SVM Training

- Update surface
- Identify Outlier

All points within acceptable error?

Yes

No

Iterator Structural Pattern
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?
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
Pattern 4: Agent and Repository

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)
Example: Compiler Optimization

- Constant folding
- Loop fusion
- Software pipelining
- Common-sub-expression elimination
- Strength-reduction
- Dead-code elimination

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
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
Pattern 5: Process Control

- **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

Source: Adapted from Shaw & Garlan 1996, p27-31.
Examples of Process Control

- **Return air**
- **Furnace**
- **Hot air**
- **Temperature sensor**
- **Gas-valve control**
- **Temperature-setting control**

User timing constraints

Process control structural pattern

- **Controller**
- **Timing constraints**
- **Circuit**

Launching transformations

Speed? Power?
Pattern 6: Model-View-Controller

- **Model**: embodies the data and “intelligence” (aka business logic) of the system
- **Controller**: captures all user input and translates it into actions on the model
- **View**: renders the current state of the model for user
Example of Model-View Controller

Control Form

View 1

View 2

User Updates Values & Presses ‘View 1’ Button

Controller Selects View

Controller

State Change

50%

30%

20%

Model

a = 50%
b = 30%
c = 20%

Extended from: Design Patterns Elements of Reusable Object-Oriented Software
Pattern 7: Layered Systems

- Individual layers are big but the interface between two adjacent layers is narrow.
- Non-adjacent layers cannot communicate directly.

Examples?
Example: ISO Network Protocol
Pattern 8: Event-based Systems

- Agents interact via events/signals in a medium
- Event manager manages events
- Interaction among agents is dynamic - no fixed connection

Examples?
Example: The Internet

- Internet is the medium
- Computers are agents
- Signals are IP packets
- Control plane of the router is the event manager
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 simulation codes to guarantee correctness of the overall simulation

- Difference with agent and repository?
  - No central repository
  - Data transfer between simulators

Examples?
Overall Computation

• 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
Remember the Analogy:
Layout of Factory Plant

- We have only talked about structure. We haven’t described computation.
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

Friday: Computational Patterns of Parallel Programming (James Demmel (UCB))
(8:45 - 10:45am)
Outline

- Intro to Kurt
- General approach to applying the pattern language
- Detail on Structural Patterns
- High-level examples of composing patterns
CBIR Application Framework

Choose Examples → Feature Extraction → Train Classifier → Exercise Classifier

New Images → Results → User Feedback

Feature Extraction

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

Build Scale-Space Representation

Select Interest Points and Support Regions

Build Descriptors

Structured Grid

Map Reduce

Structured Grid

Map Reduce

Dense Linear Algebra

"Image Feature Extraction for Mobile Processors", Mark Murphy, Hong Wang, Kurt Keutzer IISWC '09
Train Classifier:
SVM Training

Train Classifier

Update Optimality Conditions

Select Working Set,
Solve QP

MapReduce

MapReduce

Iterator Pattern
Exercise Classifier: SVM Classification

- Exercise Classifier
- Test Data
- Compute dot products
- Compute Kernel values, sum & scale
- Output

Dense Linear Algebra

MapReduce
Support-Vector Machine Mini-Framework

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

Fast support vector machine training and classification, Catanzaro, Sundaram, Keutzer, International Conference on Machine Learning 2008
Large Vocabulary Continuous Speech Recognition Poster: Chong, Yi
Work also to appear at Emerging Applications for Manycore Architecture
Architecture of Logic Optimization

Group, order tasks

Circuit representation (.db)
- Area optimization
- Timing optimization
- Power optimization

Group, order tasks

Decompose

Data
- Sensor

Group, order tasks

Graph algorithm pattern

Group, order tasks

Graph algorithm pattern
The take away

- My own experience has shown that a sound software architecture is the greatest single indicator of a software project's success.
- Software must be architectured to achieve productivity, efficiency, and correctness.
- SW architecture >> programming environments
  - >> programming languages
  - >> compilers and debuggers
  - (>>hardware architecture)
- Key to architecture (software or otherwise) is design patterns and a pattern language.
- At the highest level our pattern language has:
  - Eight structural patterns
  - Thirteen computational patterns
- Yes, we really believe arbitrarily complex parallel software can be built just from these!
What’s next …

- Friday: Computational Patterns of Parallel Programming (James Demmel (UCB))
  (8:45 - 10:45am)
Today's talk focuses only on the middle of the pattern language.
Today's talk focuses only on the middle of the pattern language.
Elements of a Pattern - 1

• **Name**
  - It must have a meaningful name useful to remember the pattern and when it is used.

• **Problem**
  - A statement of the problem ... a one-line preamble and the problem stated as a question.

• **Context**
  - The conditions under which the problem occurs. Defines when the pattern is applicable and the configuration of the system before the pattern is applied.

• **Forces**
  - A description of the relevant *forces* and constraints and how they interact/conflict with one another and with goals we wish to achieve. Defines the tension that characterizes a problem.
Elements of a Pattern - 2

- **Solution**
  - Instructions used to solve the problem. When done right, it resolves the tension defined in the forces section; flowing from the context and forces. We also define the new context for the system following application of the pattern.

- **Invariant**
  - What must be invariantly true for this pattern to work. May be stated in the form of Precondition, Invariant, Post-condition

- **Examples**
  - Examples to help the reader understand the pattern.

- **Known Uses and frameworks**
  - Cases where the pattern was used; preferably with literature references.

- **Related Patterns**
  - How does this pattern fit-in or work-with the other patterns in the pattern language.

*PLPP is the pattern language from Mattson, Sanders and Massingill
Design Patterns: “Each design 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

Structural patterns: design patterns that provide solutions to problems associated with the development of program structure

Computational patterns: design patterns that provide solutions to recurrent computational problems
Library: The software implementation of a computational pattern (e.g. BLAS) or a particular sub-problem (e.g. matrix multiply)

Framework: An extensible software environment (e.g. Ruby on Rails) organized around a structural pattern (e.g. model-view-controller) that allows for programmer customization only in harmony with the structural pattern

Domain specific language: A programming language (e.g. Matlab) that provides language constructs that particularly support a particular application domain. The language may also supply library support for common computations in that domain (e.g. BLAS). If the language is restricted to maintain fidelity to a structure and provides library support for common computations then it encompasses a framework (e.g. NPClick).
Eventually

1. Domain Experts + Application patterns & framework → End-user, application programs

2. Domain literate programming gurus (1% of the population) + Parallel patterns & programming frameworks → Application frameworks

3. Parallel programming gurus (1-10% of programmers) → Parallel programming frameworks

The hope is for Domain Experts to create parallel code with little or no understanding of parallel programming.

Leave hardcore “bare metal” efficiency layer programming to the parallel programming experts.
Today

1. Domain Experts

2. Domain literate programming gurus (1% of the population) + Application patterns & framework

3. Parallel programming gurus (1-10% of programmers) + Parallel programming frameworks

End-user, application programs

Application frameworks

Parallel programming frameworks

• For the foreseeable future, domain experts, application framework builders, and parallel programming gurus will all need to learn the entire stack.

• That’s why you all need to be here today!
# People, Patterns, and Frameworks

<table>
<thead>
<tr>
<th>People</th>
<th>Design Patterns</th>
<th>Frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Developer</strong></td>
<td>Uses application design patterns <em>(e.g. feature extraction)</em> to architect the application</td>
<td>Uses application frameworks <em>(e.g. CBIR)</em> to develop application</td>
</tr>
<tr>
<td><strong>Application-Framework Developer</strong></td>
<td>Uses programming design patterns <em>(e.g. Map/Reduce)</em> to architect the application framework</td>
<td>Uses programming design patterns <em>(e.g. MapReduce)</em> to develop the programming framework</td>
</tr>
</tbody>
</table>