Disclaimer

READ THIS ... its very important

- The views expressed in this talk are those of the speakers and not their 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 we say anything really stupid, it’s our fault ... don’t blame our collaborators.

---

**Parallel Computing: SW Architecture, design patterns and surviving the many core revolution**

*Part 1: Introduction and background*

Tim Mattson (Intel Labs) and Michael Wrinn (Intel SSG)

---

**Why patterns? The premise:**

- Difficulties in facing the challenge of developing parallel software are a symptom of underlying weakness in our abilities to:
  - Architect software
  - Re-use implementation approaches

- If the solution to the parallel programming doesn’t have implications for all types of software design, then it’s not a viable solution to parallel programming!

---

**Outline**

- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

---

**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 9, A Pattern Language, Christopher Alexander
- Alexander’s 253 (civl) architectural patterns range from the creation of cities (21, 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!!
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 of the 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 example 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).

Computational Patterns

- Derived from the Dwarfs in "The Berkeley View" (Asanovic et al.)
- Dwarfs form our key computational patterns

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

OPL Pattern Language (Keutzer & Mattson 2010)

To get frameworks right ... start with an understanding of software architecture

PLPP: Pattern language of Parallel Programming

© Kurt Keutzer 2012
Our Pattern Language

Applications

Structural Patterns

Computational Patterns

Parallel Algorithm Strategy Patterns

Implementation Patterns

Execution Patterns

Outline

Introduction to Patterns and Our Pattern Language

Architecting Parallel Software: structure & computation

Structural Patterns

Computational Patterns

Parallel (Algorithm, Implementation, Execution) Patterns

Examples

Summary

Identify the SW Structure

Identify Key Computations

Analogy: Layout of Factory Plant

Analogy: Machinery of the Factory

Structural Patterns

Pipe-and-Filter

Agent-and-Repository

Process-Control

Event-Based/Implicit-Invocation

Puppeteer

Model-View-Controller

Iterative-Refinement

Map-Reduce

Layered-Systems

Arbitrary-Static-Task-Graph

These define the software structure but do not describe what is computed.

These define the key computations, but do not describe how they are implemented.
Architecting the Whole Application

- SW Architecture of Large-Vocabulary Continuous Speech Recognition
- Raises appropriate issues like scheduling, latency, throughput, workflow, resource management, capacity etc.

Outline

- Introduction to Patterns and Our Pattern Language
- Architecting Parallel Software
  - Structural Patterns – 2 examples (of 10)
    - Pipe and filter
    - Map Reduce
  - Computational Patterns
  - Parallel (Algorithm, Implementation, Execution) Patterns
- Examples
- Summary

Structural Pattern: Pipe and Filter

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

Examples of pipe and filter

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

Examples of Map Reduce

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

Examples

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

- Support-vector machines (ML)
  - Map to evaluate distance from the frontier
  - Reduce to find the greatest outlier from the frontier
Computational Pattern: Spectral Methods

“Spectral Methods” are a broad class of numerical algorithms for solving PDEs, but notions of Spectral Analysis (i.e. convenient changes of basis) are important in every application area.

In Magnetic Resonance Imaging (MRI), images are collected in “k-space” -- i.e. an MRI scan produces a Fourier Domain image.

Fourier and Wavelet representations are different spectral analyses that expose different properties of images convenient for solving our problems.

Spectral Methods Pattern: Fast Transforms

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

$$D_0 = \sum_{k=0}^{N-1} a_k w_0^k$$

$$P = \begin{bmatrix}
1 & 1 & 1 & \cdots & 1 \\
1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\
1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)}
\end{bmatrix}$$

$$F_{x} \rightarrow \begin{bmatrix}
L_{x,0} & L_{x,1} & L_{x,2} & \cdots & L_{x,N-1} \\
R_{x,0} & R_{x,1} & R_{x,2} & \cdots & R_{x,N-1}
\end{bmatrix} \begin{bmatrix}
F_{x,0} \\
F_{x,1} \\
F_{x,2} \\
\vdots \\
F_{x,N-1}
\end{bmatrix}$$

Structural Patterns and Parallelism

- Structural patterns are “above the concerns of parallelism” – they may be used for either serial or parallel implementations.
- When we attempt to parallelize a software architecture, the description in structural patterns gives us only a modest amount of parallelism.
  - Map reduce may be the only exception among the structural patterns.
- Therefore we need a new perspective on parallelizing the architecture – i.e. new patterns.

OPL: Parallel Algorithm Strategy

Applications

- Structural Patterns
- Computational Patterns

Parallel Algorithm Strategy Patterns

Implementation Patterns

Execution Patterns
Parallel Algorithm Strategy Patterns

These patterns define high-level strategies to exploit concurrency within a computation for execution on a parallel computer. They address the different ways concurrency is naturally expressed within a problem/application.

How does the software architecture map onto parallel algorithms?

<table>
<thead>
<tr>
<th>Algorithm Strategy Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Parallelism</td>
</tr>
<tr>
<td>Recursive splitting</td>
</tr>
<tr>
<td>Task Parallelism</td>
</tr>
<tr>
<td>Data Parallelism</td>
</tr>
<tr>
<td>Pipeline</td>
</tr>
<tr>
<td>Discrete Event</td>
</tr>
<tr>
<td>Geometric Decomposition</td>
</tr>
<tr>
<td>Speculation</td>
</tr>
</tbody>
</table>

Implementation Strategy Patterns

These are the structures that are realized in source code to support (a) how the program itself is organized and (b) common data structures specific to parallel programming.

How do parallel algorithms map onto source code in a parallel programming language?

<table>
<thead>
<tr>
<th>Implementation Strategy Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMPD</td>
</tr>
<tr>
<td>Master/worker</td>
</tr>
<tr>
<td>Strict data par</td>
</tr>
<tr>
<td>Task-Queue</td>
</tr>
<tr>
<td>Fork/Join</td>
</tr>
<tr>
<td>Graph partitioning</td>
</tr>
<tr>
<td>Loop-Par.</td>
</tr>
<tr>
<td>Program structure</td>
</tr>
<tr>
<td>Shared-Queue</td>
</tr>
<tr>
<td>Distributed-Array</td>
</tr>
<tr>
<td>Shared-Queue</td>
</tr>
<tr>
<td>Distributed-Array</td>
</tr>
<tr>
<td>Data structure</td>
</tr>
</tbody>
</table>

Parallel Execution Patterns

These are the approaches often embodied in a runtime system that supports the execution of a parallel program.

How is the source code realized as an executing program running on the target parallel processor?

<table>
<thead>
<tr>
<th>Parallel Execution Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIMD</td>
</tr>
<tr>
<td>SIMD</td>
</tr>
<tr>
<td>Thread Pool</td>
</tr>
<tr>
<td>Task-Graph</td>
</tr>
<tr>
<td>Task-Graph</td>
</tr>
<tr>
<td>Data Flow</td>
</tr>
<tr>
<td>Digital Circuits</td>
</tr>
<tr>
<td>Msg Pass</td>
</tr>
<tr>
<td>Collective Comm</td>
</tr>
<tr>
<td>Mutual exclusion</td>
</tr>
<tr>
<td>Pt-2-pt sync</td>
</tr>
<tr>
<td>Collective sync</td>
</tr>
<tr>
<td>Trans Mem</td>
</tr>
<tr>
<td>Coordination</td>
</tr>
</tbody>
</table>

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - SVM classifier
- Summary
Support Vector Machine Classifier

Feature Extraction

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

Feature Extraction

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

Train Classifier: SVM Training

- SVM Training
  - Update Optimality Conditions
  - Select Working Set, Solve QP

Exercise Classifier: SVM Classification

- SVM Classification
  - Compute dot products
  - Compute Kernel values, sum & scale
  - Output

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - Large vocabulary continuous speech recognition
- Summary

Support-Vector Machine Mini-Framework

- Support-Vector Machine Framework used to achieve:
  - 9-35x speedup for training
  - 81-138x for classification
  - 1100 downloads since release


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


© Kurt Keutzer
LVCSR Software Architecture

- Inference engine based system
  - Used in Sphinx (CMU, USA), HTK (Cambridge, UK), and Julius (CMRC, Japan) [10,15,9]
  - Modular and flexible setup
    - Shown to be effective for Arabic, English, Japanese, and Mandarin

Key computation: HMM Inference Algorithm

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

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

Viterbi Algorithm

- One filter per state
- Observation probability computation

Inference Engine in LVCSR

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

Parallelism in the inference engine:

0. Gather operand

1. \[ P(x|s) \]

2. \[ m[t][s] \]

Each Filter is a Map Reduce

0. Gather operands

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

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

Each Filter is Map Reduce

1. observation probability computation

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

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

Each Filter is Map Reduce
2. graph traversal computation

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

\[ m[t][s] = \max_{s' \in S} m[t-1][s'] \cdot P(s|s') \cdot P(s_t|s) \]

LVCSR Software Architecture

- **Inference Engine**
  - Beam Search Iterations
  - Graphical Model
  - Dynamic Programming

- **Word Sequence**

Speech Recognition with HMM

<table>
<thead>
<tr>
<th>Observation</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wreck</td>
<td>State 1</td>
</tr>
<tr>
<td>a</td>
<td>State 2</td>
</tr>
<tr>
<td>nice</td>
<td>State 3</td>
</tr>
<tr>
<td>beach</td>
<td>State 4</td>
</tr>
<tr>
<td>Recognize</td>
<td>State 5</td>
</tr>
</tbody>
</table>

Interpretation:
- Wreck
- a nice beach speech

Speech Recognition Results

- Input: Speech audio waveform
- Output: Recognized word sequences
- Achieved 11x speedup over sequential version
- Allows 3.5x faster than real time recognition
- Our technique is being deployed in a hot-line call-center data analytics company
- Used to search content, track service quality and provide early detection of service issues

Recognize, Enhance, Transcribe, Store

Scalable HMM based Inference Engine in Large Vocabulary Continuous Speech Recognition, Kisu Yu, Jialei Cheng, Youngsun H. Baek, Giovanni Cristani, Christopher Hughes, Wonyong Sung and Kurt Keutzer, IEEE Signal Processing Magazine, March 2010

Video Game Physics

- Modern 3D Video Game Engines are simulations of a virtual world
  - Simulation of physics just realistic enough to be convincing. Physical accuracy (e.g. fidelity to CFD equations) is secondary to visual appearance and compute budget.
  - "Pre-Computed" physics => animations
- Scenery: E.G. Buildings, Trees, Terrain, ... Can collide with non-Scenery objects, but don’t move Usually immutable -- appearance is pre-computable
- Physical Objects: E.G. vehicles, characters, projectiles, ... Collide, explode, move: fully dynamic Rigid-body simulations, Vehicle simulation, "Ragdoll Corpses"
- Non-Physical Objects: E.G. lights, smoke plumes, Most "Physical" computations in games are actually quick, cheap hacks that give the visual appearance of physical accuracy

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - Game development, as presented in:
  - UC Berkeley undergraduate course CS194 Engineering Parallel Software, Fall 2010
- Summary

From research concept to full undergraduate class: 2 years

© Kurt Keutzer
Other sources of computation

- **Animation**: Updating the visual state and physical configuration of characters and objects: crucial for realism, closely coupled with AI.
- "**Artificial Intelligence**" (AI): Mostly scripts and ad-hoc decision trees, but physical queries (am I close to something interesting?) and graph-searching / path planning.
- **Damage and Destruction**: E.g. vehicles, buildings, characters can be destroyed, burned, modified. I.E. produce physical objects!
- **Audio**: The many sources of sound in a game’s scene must be processed and combined according to occlusion, distance from player, etc.
- **Input System**: Currently, very trivial ... but with the introduction of accelerometers, video cameras, etc. could get very interesting.

Architecting a Video Game

- The Game Engine’s purpose is to maintain, update, and display the state of the Virtual World it is simulating.
- Each stateful sub-system.
- How can we organize the structure of this software?

Anti-Solution: Coarse Multithreading

- Assign each sub-module to a different CPU Thread
  - Was the first pass approach of Game Engine implementers
- Subsystem interactions become Mutual Exclusions on Shared Data
  - In Testing, Verification and Debugging, you are at the mercy of the OS’s non-deterministic, non-repeatable thread scheduler
- Load Balancing: the "Physical Simulation" thread will have much more work than the "Scripting" thread

Architecture, one choice: MVC Pattern

- **Model View Controller Structural Pattern**:
  - Model: maintain the system’s state, update "observers"
  - View: Renders the model for interaction with User
  - Controller: Initiates responses to Input by modifying the Model

Module-Level Parallelism

- Executing multiple sub-modules in parallel present problems:
  - Modules share data: some have read-only access to state that other modules will update
  - We need a way of managing access to the Data. Suggestions?

Agent and Repository

- Agent and Repository pattern is an approach to assigning modules responsibility in accessing a shared data structure
  - Repository is responsible for partitioning of the data and scheduling of the agents
  - Agents execute when Repository Manager allows them to
Puppeteer Pattern

- An alternative: The Puppeteer. Sub-modules are responsible for the management of their own data (no central repository)
- Repository Manager owned data in A&R
- Puppeteer only owns the channels of communication
- Or "Baked" into the puppeteer implementation

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Parallel (Algorithm, Implementation, Execution) Patterns
- Real world examples
  - Pediatric MRI
  - Summary

Compelling Application: Fast, Robust Pediatric MRI

Pediatric MRI is difficult:
- Children cannot sit still, breathhold
- Low tolerance for long exams
- Anesthesia is costly and risky

Like to accelerate MRI acquisition
- Advanced MRI techniques exist, but require data- and compute-intensive algorithms for image reconstruction

Reconstruction must be fast, or time saved in accelerated acquisition is lost in computing reconstruction
- Slow reconstruction times are a non-starter for clinical use

Domain Experts and State-of-the-Art Algorithms

Collaboration with MRI Researchers:
- Miki Lustig, Ph.D., Berkeley EECS
- Marc Alley, Ph.D., Stanford EE
- Shreyas Vasanawala, M.D./Ph.D., Stanford Radiology

Advanced MRI: Parallel Imaging and Compressed Sensing to dramatically reduce MRI image acquisition time

\[
\text{minimize } ||Wx||_1 \\
\text{s.t } F_0x = y, \\
||Gx - x||_2 < \epsilon
\]
**Game-Changing Speedup**

100X faster reconstruction
Higher-quality, faster MRI

This image: 8 month-old patient with cancerous mass in liver
- 256 x 84 x 154 x 8 data size
- Serial Recon: 1 hour
- Parallel Recon: 1 minute

Fast enough for clinical use
- Software currently deployed at Lucile Packard Children’s Hospital for clinical study of the reconstruction technique

**Summary**

What’s the point?
- Parallel design patterns give a new powerful viewpoint to programmers
- Enable us to disentangle the big fuzzy ball of yarn of computation
  - add 20 IQ points to our problem solving (as per Alan Kay)
- Our Pattern language helps you to write good parallel code

**Software Design Patterns in education: lessons from history?**

<table>
<thead>
<tr>
<th>Early days OO</th>
<th>Now</th>
<th>Future</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage: specialists only, Mainstream regards with indifference or anxiety.</td>
<td>Usage: specialists only, Mainstream regards with indifference or anxiety.</td>
<td>Usage: widespread, key concepts actually deployed.</td>
</tr>
</tbody>
</table>

1994

2012

6/6/2012