Assumption #1: How not to develop parallel code

Initial Code → Profiler → Performance profile

Re-code with more threads → Not fast enough

Fast enough → Ship it

Lots of failures
N PE’s slower than 1
### 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>average</td>
<td>1.00</td>
<td>1.0011</td>
<td>1.0044</td>
<td>1.0049</td>
<td>1.0046</td>
<td>1.0046</td>
</tr>
</tbody>
</table>

### Assumption #2: Nor this

- **Initial Code**: Super-compiler
- **Performance profile**: Not fast enough
- **Tune compiler**: 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.

Assumption #3: This won’t help either

- Code in new cool language
- Re-code with cool language
- Not fast enough
- Fast enough
- Ship it

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

---

So What’s the Alternative?
Principles of SW Design

- After 15 years in industry, at one time overseeing the technology of 25 software products, my two best principles to facilitate good software design are:
  - Use of modularity
  - Definition of invariants
- Modularity helps:
  - Architect: Makes overall design sound and comprehensible
  - Project manager:
    - As a manager I am able to comfortably assign different modules to different developers
    - I am also able to use module definitions to track development
  - Module implementors: As a module implementor I am able to focus on the implementation, optimization, and verification of my module with a minimum of concern about the rest of the design
  - Identify invariants and key computations

Non-Principles of SW Design

- What’s life like without modularity?
  - Spaghetti code
  - Wars over the interpretation of the specification
  - Waiting on other coders
  - Wondering why you didn’t touch anything and now your code broke
  - Hard to verify your code in isolation, and therefore hard to optimize
  - Hard to parallelize without identifying key computations

- Modularity will help us obviate all these
  - Parnas, “On the criteria to be used on composing systems into modules,” CACM, December 1972.
  - But modularity alone is not enough, because …
Modularity is important .... But ...

Pop quiz: Is software more like?

a) A building  

b) A factory

What computations we do is as important than how we do them ....
Outline

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

Identify the SW Structure

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

These define the structure of our software but they do not describe what is computed.
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
  Analogous to the design of an entire manufacturing plant

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

Connectors are where the communication happens
Inventory of Structural Patterns

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

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

Pattern 1: Pipe and Filter

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

- 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

Synchronize results of iteration

Exit condition met?

Yes

No

Initialize condition

iterate

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

- Update surface
- Identify Outlier
- Iterate

All points within acceptable error?
- Yes
- No

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/(minimax)

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

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
  - Linear Algebra
  - Spectral Methods
  - Dynamic programming
- Examples
- Summary
We build arbitrarily complex computations out of these thirteen computational patterns.

**CP1: Linear Algebra**

- **Vector Space**: A set closed under + has identity and inverse elements, scalar multiplication
- **Linear Map**: Operator \( T \) on vectors \( u, v \), scalar \( \alpha \) s.t.
  \[ T(u + v) = Tu + Tv \text{, and } T(\alpha v) = \alpha T(v) \]
- **Matrix**: An \( m \times n \) array of numbers representing a linear map from \( \mathbb{R}^n \) to \( \mathbb{R}^m \)
- **Linear Equations**: \( Ax = b \)
- **Eigenvalues/vectors**: \( Ax = \lambda x \)
Basic Linear Algebra Subroutines (BLAS)

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

<table>
<thead>
<tr>
<th>Level</th>
<th>Example</th>
<th># mem refs</th>
<th># flops</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xAXPY: y = y + αx</td>
<td>3n</td>
<td>2n^1</td>
<td>2/3</td>
</tr>
<tr>
<td>2</td>
<td>xGEMV: y = y + Ax</td>
<td>n^2</td>
<td>2n^2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>xGEMM: C = C + AB</td>
<td>4n^2</td>
<td>2n^3</td>
<td>n/2</td>
</tr>
</tbody>
</table>

Linear Algebra Resources

www.netlib.org/lapack
www.netlib.org/scalapack
gams.nist.gov
www.netlib.org/templates
www.cs.utk.edu/~dongarra/etemplates
"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!
Spectral Methods Pattern: Libraries

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

CP3: Dynamic Programming

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

Diagram: Dynamic Programming:
- Best cover for this match uses best covers for x, y, z
- Choose least cost tree-cover at root
- Best cover for this match uses best covers for p, z
Mapping a circuits into a Cell Library

Example of Optimal Tree Covering
Speech Recognition

Observations

Time

Wreck

a

nice

beach

Recognize

speech

Interpretation

20
Outline

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

Support Vector Machine Classifier

Feature Extraction

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

```
Build Scale-Space Representation
   =
Structured Grid

Select Interest Points and Support Regions
   +
Map Reduce
   =
Dense Linear Algebra

Build Descriptors
   +
Map Reduce
   =
Structured Grid
```

"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
   +
Iterate
```

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

- Test Data
- SV
- 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
  - 1100 downloads since release

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

- Computational IOU: Must solve constrained L1 minimization

\[
\begin{align*}
\text{minimize} & \quad \|Wx\|_1 \\
\text{s.t.} & \quad F_\Omega x = y, \\
& \quad \|Gx - x\|_2 < \varepsilon
\end{align*}
\]

SW architecture of image reconstruction

Pipe and Filter
- Data Parallelism / Fourier Transforms

Fork-Join
- Linear Alg.

Data Parallelism / Fourier Transforms

Fork Join

Data Parallelism / Fourier Transforms

Iterative POCS Algorithm:
1. Apply SPIRIT Operator:
   \[ x_c \leftarrow \sum g_{cj} \cdot x_j \]
2. Wavelet Soft-Thresholding
   \[ x \leftarrow W S_\lambda \{W^* x\} \]
3. Fourier-space projection
   \[ x \leftarrow F(P^T y + P^T P_c F^* x) \]
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

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
  - CBIR
  - MRI
  - Speech Recognition
- Summary
- 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
Key computation: HMM Inference Algorithm

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

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

Viterbi Algorithm

Legends:
- A State
- An Observation
- \(P(x_t | s_t)\)
- \(m[t-1][s_{t-1}]\)

Markov Condition:

\[
m[t][s_t] = \max_{s_{t-1}} P(x_0, x_1, \ldots, x_t, s_0, \ldots, s_{t-1}, s_t)
\]

Inference Engine in LVCSR

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

Parallelism in the inference engine:
Speech Recognition with HMM

Speech Recognition Results

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

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

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

Scalable HMM based Inference Engine in Large Vocabulary Continuous Speech Recognition, Kisun You, Jike Chong, Youngmin Yi, Ekaterina Gonina, Christopher Hughes, Wonyong Sung and Kurt Keutzer, IEEE Signal Processing Magazine, March 2010
Extended audio-only speech recognition framework to enable audio-visual speech recognition (lip reading)

Achieved a 20x speedup in application performance compared to a sequential version in C++

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

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

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
- Summary
Other Interesting Results

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

Summary

- The key to productive and efficient parallel programming is creating a good software architecture – a hierarchical composition of:
  - Structural patterns: enforce modularity and expose invariants
    - I showed you three – seven more will be all you need
  - Computational patterns: identify key computations to be parallelized
    - I showed you three – ten more will be all you need
  - Orchestration of computational and structural patterns creates architectures which greatly facilitates the development of parallel programs:
    - I showed you three – there are many more

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

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