Architecting Parallel Software with Patterns

Kurt Keutzer
the PALLAS group,
Michael Anderson, Bryan Catanzaro, Jike Chong,
Katya Gonina, Dorothea Kolossa, Chao-Yue Lai,
Mark Murphy, David Sheffield, Bor-Yiing Su,
Narayanan Sundaram
with thanks to Tim Mattson

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

© Kurt Keutzer
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 → Ship it

Tune compiler

Not fast enough: Fast enough

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
- Profiler
- Performance profile
- Fast enough
- Re-code with cool language
- Not fast enough
- Ship it
- After 200 parallel languages where’s the light at the end of the tunnel?
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.
Modularity is important .... But ...

Pop quiz: Is software more like?

a) A building
b) A factory

![Building and Factory Images]

© Kurt Keutzer

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

<table>
<thead>
<tr>
<th>Apps</th>
<th>Dwarves</th>
<th>Embedded</th>
<th>SPEC</th>
<th>DB</th>
<th>Games</th>
<th>ML</th>
<th>HPC</th>
<th>CAD</th>
<th>Health</th>
<th>Image</th>
<th>Speech</th>
<th>Music</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backtrack / B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circuits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Prog.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monte Carlo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N-Body</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Kurt Keutzer
Putting computation and structure together:
- We believe the key to productively building efficient and correct parallel software is software architecture

A software architecture is a hierarchical composition of:
- Computational patterns – the atoms
- Structural patterns – the molecular bonds

This software architecture naturally gives:
- Modularity
  - Efficient management
  - Efficient implementation
  - Efficient verification
- Identifies key computations, invariants, and interfaces

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

© Kurt Keutzer
### Identify Key Computations

<table>
<thead>
<tr>
<th>Apps</th>
<th>Dwarves</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Embed</td>
</tr>
<tr>
<td></td>
<td>SPEC</td>
</tr>
<tr>
<td></td>
<td>DB</td>
</tr>
<tr>
<td></td>
<td>Games</td>
</tr>
<tr>
<td></td>
<td>ML</td>
</tr>
<tr>
<td></td>
<td>HPC</td>
</tr>
<tr>
<td></td>
<td>CAD</td>
</tr>
<tr>
<td></td>
<td>Health</td>
</tr>
<tr>
<td></td>
<td>Image</td>
</tr>
<tr>
<td></td>
<td>Speech</td>
</tr>
<tr>
<td></td>
<td>Music</td>
</tr>
<tr>
<td></td>
<td>Browser</td>
</tr>
</tbody>
</table>

- 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
• 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 pattern 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

© Kurt Keutzer
Filter 1

Pattern 1: Pipe and Filter

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

• Pipes embody communication

Filter 2

Filter 3

Filter 4

Filter 5

Filter 6

Filter 7

May have feedback

Examples?

Examples of pipe and filter

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

Program

Scan Program

Build Internal Representation

Optimize Program

Generate Code

Object code

Compiler

New Images

Choose Examples

Feature Extraction

Train Classifier

Exercise Classifier

Results

User Feedback

Image Retrieval System

Logic optimizer

Scan Netlist

Build Data model

Optimize circuit

Optimize netlist

© Kurt Keutzer
Pattern 2: Iterator Pattern

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

Synchronize results of iteration

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
Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
  - Linear Algebra
  - Spectral Methods
  - Dynamic programming
- Examples
- Summary

Inventory of Key Computations

<table>
<thead>
<tr>
<th>Apps</th>
<th>Dwarves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph Algorithms</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Graphical Models</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Backtrack / B&amp;B</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Finite State Machine</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Circuits</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Dynamic Prog.</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Unstructured Grid</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Structured Grid</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Dense Matrix</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Spectral (FFT)</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>Monte Carlo</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
<tr>
<td>N-Body</td>
<td>Embedded SPEC Databases Games ML HPC CAD Health Image Speech Music Browser</td>
</tr>
</tbody>
</table>

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. $T(u + v) = Tu + Tv$, 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: $y = y + \alpha x$</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>
CP2: Spectral Methods Pattern: MRI Reconstruction

- "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!

$$W_N = \sum_{k=0}^{N-1} x_k \omega^k$$

$$F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \omega & \omega^2 & \omega^3 & \omega^4 \\ \omega^2 & \omega^4 & \omega^6 & \omega^8 \\ \omega^3 & \omega^6 & \omega^9 & \omega^{12} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -\frac{1}{2} & -1 & \frac{1}{2} \\ 1 & -1 & 1 & -1 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

$$F_N = \begin{bmatrix} I_{n/2} & D_{n/2} \\ I_{n/2} & -D_{n/2} \end{bmatrix} = \begin{bmatrix} F_{n/2} \text{ Reuse} \\ F_{n/2} \text{ Odd} \end{bmatrix}$$

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 of Dynamic Programming](image1)

Mapping a circuits into a Cell Library

![Diagram of Mapping into Cell Library](image2)
Example of Optimal Tree Covering

INV 11 + 2 = 13
NAND2 2 + 6 + 3 = 11
INV 2
NAND2 3 + 3 = 6
AOI21 4 + 3 = 7
NAND2 3

code generation in compilers

LOAD r1 ← M[fp + a]
ADD r2 ← r1 + 4
MUL r3 ← r2 × r1
LOAD r2 ← M[fp + x]
ADD r4 ← r1 + r2
ADD r5 ← fp + x
STORE M[r4 + 0] ← r3
MOVE M[r5] ← M[r2]
Speech Recognition

Observations

Time

Speech Model States

Wreck a nice beach
Recognize

Speech

m

Outline

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

© Kurt Keutzer
Support Vector Machine Classifier

- Choose Examples
- Feature Extraction
- Train Classifier
- Exercise Classifier
- 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

Dense Linear Algebra

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

© Kurt Keutzer
Train Classifier: SVM Training

- Train Classifier
- Update Optimality Conditions
- Select Working Set, Solve QP

MapReduce

Exercise Classifier: SVM Classification

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

Dense Linear Algebra

MapReduce

© Kurt Keutzer
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

Outline

- Architecting Parallel Software
- Structural Patterns
- Computational Patterns
- Examples
  - CBIR
  - MRI
  - Speech Recognition
- 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- 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

\[
\text{minimize } \| Wx \|_1 \\
\text{s.t. } F_{\Omega}x = y, \\
\| Gx - x \|_2 < \varepsilon
\]
**SW architecture of image reconstruction**

Pipe and Filter
- Data Parallelism / Fourier Transforms
- Fork-Join
  - Linear Alg.
  - Linear Alg.
- Data Parallelism / Fourier Transforms
- Fork Join
- Data Parallelism / Fourier Transforms

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

- Iterative Refinement / Spectral Method
- Data Parallelism / Convolutions
- Data Parallelism / Wavelet xforms
- Data Parallelism / Fourier xforms

---

**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

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
Inference Engine

Beam Search Iterations

Active State Computation Steps

Graphical Model

Dynamic Programming

Iterative Refinement

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

Markov Condition:

\[ m[t][s_t] = \max_{s_{t-1}} P(x_0, x_1, \ldots, x_t|s_0, s_1, \ldots, s_t) \]

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

Parallelism in the inference engine:

0. Gather operand
1. $P(x_i|s_j)$
2. $m[t][s_j]$
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

Multi-media Speech Recognition

- Prof. Dorothea Kolossa
  Speech Application Domain Expert
  Technische Universität Berlin

- Extended audio-only speech recognition framework to enable audio-visual speech recognition (lip reading)

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

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

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

© Kurt Keutzer
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
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

CS194: Engineering Parallel Software: Fall 2010

© Kurt Keutzer