Hands On: Multimedia Methods for Large Scale Video Analysis (Lecture)

Dr. Gerald Friedland, fractor@icsi.berkeley.edu
Today

- Answers to Questions
- How to estimate resources for large data projects
  - Some basic issues
  - Factors to take into account
  - Different methods to estimate
  - Example
FAQ

• How is the grade computed?

• 15% Mid-Term Exam
• 15% Final Exam
• 70% Project
FAQ

• What is graded in the project?
  • Final Presentation
  • Project Report (details to follow)
Some Basics
Some Basics

• Why can’t we automatize the
Some Basics

- Why can’t we automatize the estimation of resources for large
Some Basics

- Why can’t we automatize the estimation of resources for large data projects?
Some Basics

Why can’t we automatize the estimation of resources for large data projects?

Halting problem
Some Basics

- Why can’t we automatize the estimation of resources for large data projects?

Halting problem
undecidable!!
Some Basics

- Why can’t we automatize the estimation of resources for large data projects?

Halting problem undecideable!!
More Basics
Why can’t we just parallelize everything?
Why can’t we just parallelize everything?

Amdahl’s Law!
More Basics

Why can’t we just parallelize everything?

Amdahl’s Law!

Amdahl’s Law

\[ S(N) = \frac{1}{(1 - P) + \frac{P}{N}}. \]

where:
S is the speed-up expected
P is the unparallelizable portion
N is the number of CPUs
Amdahl’s Law

\[ P_{\text{estimated}} = \frac{1}{SU} - \frac{1}{NP} - 1. \]

where:

- SU is the speed-up measured
- NP is the number of CPUs
Amdahl’s Law
Amdahl’s Law
Amdahl’s Law

• Intuition: Unparallelizeable portion dominates runtime.
Amdahl’s Law

- Intuition: Unparallelizable portion dominates runtime.

=> Diminishing returns when adding more CPUs.
Factors to take into Account when estimating runtime
Factors to take into Account when estimating runtime

- CPU usage (number of instructions)
Factors to take into Account when estimating runtime

- CPU usage (number of instructions)
- I/O usage
Factors to take into Account when estimating runtime

• CPU usage (number of instructions)
• I/O usage
• Memory requirements
Factors to take into Account when estimating runtime

- CPU usage (number of instructions)
- I/O usage
- Memory requirements
- External factors
Factors to take into Account when estimating runtime

- CPU usage (number of instructions)
- I/O usage
- Memory requirements
- External factors
- The human factor
CPU Usage
CPU Usage

How many instruction does the program need per byte of input data?
CPU Usage

- How many instruction does the program need per byte of input data?
- Similar to $O(n)$ notation of theoretical computer science
CPU Usage

- How many instruction does the program need per byte of input data?
- Similar to O(n) notation of theoretical computer science
- Problem: Machine learning algorithms converge! Runtime ~ “Entropy” of input
I/O Usage
I/O Usage

- I/O might be:
  - memory access
  - storage (HD) access
  - network access
I/O Usage

- I/O might be:
  - memory access
  - storage (HD) access
  - network access

- Orders of magnitude slower than CPU access!
I/O Usage

Figure 1.5  How a modern computer system works.

Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Eighth Edition", Chapter 1
I/O Usage

Figure 1.7 A dual-core design with two cores placed on the same chip.
I/O Usage
I/O Usage

• How do you store and access the central data input?
I/O Usage

- How do you store and access the central data input?
- How do you access and store intermediate files?
I/O Usage

- How do you store and access the central data input?
- How do you access and store intermediate files?
- How do you write out (debugging) output.
I/O Usage

• How do you store and access the central data input?
• How do you access and store intermediate files?
• How do you write out (debugging) output.
• How many I/O operations will it be?
I/O Usage

• How do you store and access the central data input?
• How do you access and store intermediate files?
• How do you write out (debugging) output.
• How many I/O operations will it be?
• Using which device latencies?
Memory Requirements
Memory Requirements

- How can the cache be used most efficiently?
Memory Requirements

- How can the cache be used most efficiently?
- How much memory is available per CPU?
Memory Requirements

• How can the cache be used most efficiently?
• How much memory is available per CPU?
• Is the memory actually physical or virtual? Avoid Thrashing!
External Factors
External Factors

- How many CPUs are available at each time?
External Factors

- How many CPUs are available at each time?
- Are the CPUs of the same type?
External Factors

• How many CPUs are available at each time?
• Are the CPUs of the same type?
• Are special purpose processors available (GPUs)?
External Factors

- How many CPUs are available at each time?
- Are the CPUs of the same type?
- Are special purpose processors available (GPUs)?
- How expensive is data transfer?
External Factors

- How many CPUs are available at each time?
- Are the CPUs of the same type?
- Are special purpose processors available (GPUs)?
- How expensive is data transfer?
- Is the system functioning reliably?
Human Factors
Do you still need to experiment?
Human Factors

• Do you still need to experiment?
• Are there possibly bugs?
Human Factors

- Do you still need to experiment?
- Are there possibly bugs?
- Can the algorithm be continued?
Human Factors

- Do you still need to experiment?
- Are there possibly bugs?
- Can the algorithm be continued?
- Do I need to supervise the execution?
Human Factors

• Do you still need to experiment?
• Are there possibly bugs?
• Can the algorithm be continued?
• Do I need to supervise the execution?
• When does my batch finish? Do I need to logout in the middle of the night?
Manual Estimation
Manual Estimation

- Check loops and nested loops:
  for, while
Manual Estimation

- Check loops and nested loops:
  for, while

Nesting over input determines order of magnitude!
Manual Estimation

- Check loops and nested loops: 
  for, while
  Nesting over input determines order of magnitude!
- Find memory allocations: 
  new(), malloc(), +
Manual Estimation

- Check loops and nested loops: for, while
  Nesting over input determines order of magnitude!
- Find memory allocations: new(), malloc(), +
- Find I/O commands: read(), write()
Problem

```java
String readFile(java.io.FileReader fileReader) throws java.io.IOException {
    java.io.BufferedReader br=new java.io.BufferedReader(fileReader);
    String s="";
    String line;
    while (null!=(line=br.readLine())) s+=line+"\n";
    return s;
}
```
Problem

- Manual estimation hard and often counterintuitive!

Example:

```java
String readFile(java.io.FileReader fileReader) throws java.io.IOException {
    java.io.BufferedReader br=new java.io.BufferedReader(fileReader);
    String s="";
    String line;
    while (null!=(line=br.readLine())) s+=line+"\n";
    return s;
}
```
Problem
Problem

• Reading in the ASCII Version of Moby Dick with 17525 lines of code and 69 char per line (1.1 MB, not BIGDATA)…
Problem

• Reading in the ASCII Version of Moby Dick with 17525 lines of code and 69 char per line (1.1 MB, not BIGDATA)...

• ...shuffles around Gigabytes!
Formal Methods
Formal Methods

- Similarity templates
Formal Methods

- Similarity templates
- Rough Set-based methods
Formal Methods

- Similarity templates
- Rough Set-based methods
- Queueing theory
Formal Methods

- Similarity templates
- Rough Set-based methods
- Queueing theory
- Autotuning
Formal Methods

- Similarity templates
- Rough Set-based methods
- Queueing theory
- Autotuning

Problem: Usually limited due to assumptions, might err as well...
Inductive Algorithm Design
Inductive Algorithm Design

• Show algorithm works on small set
Inductive Algorithm Design

- Show algorithm works on small set
- Show error doesn’t get worse when adding more data
Inductive Algorithm Design

- Show algorithm works on small set
- Show error doesn’t get worse when adding more data
- Deduct that it will work for arbitratry data sizes
Problem: Non-Linear Scale

PARDORA Query Performance

- time (seconds)
- Number of songs
- Total Time
Optimization: General Approach
Optimization: General Approach

- Use spiral development scheme:
Optimization: General Approach

• Use spiral development scheme:
  • Estimate computational needs on smaller scale.
Optimization: General Approach

- Use spiral development scheme:
  - Estimate computational needs on smaller scale.
  - Verify empirically.
Optimization: General Approach

- Use spiral development scheme:
  - Estimate computational needs on smaller scale.
  - Verify empirically.
  - If verification OK, go larger scale and repeat
Optimization: General Approach

- Use spiral development scheme:
  - Estimate computational needs on smaller scale.
  - Verify empirically.
  - If verification OK, go larger scale and repeat
  - If not: debug estimation!
How to debug estimation
How to debug estimation

• Find the bottlenecks
How to debug estimation

• Find the bottlenecks
• Prioritize worst bottleneck first
How to debug estimation

- Find the bottlenecks
- Prioritize worst bottleneck first

- Problem: Diminishing Returns!
Fighting Bottlenecks
Fighting Bottlenecks

• Is there a faster algorithm?
Fighting Bottlenecks

• Is there a faster algorithm?
• Would an approximation do it?
Fighting Bottlenecks

• Is there a faster algorithm?
• Would an approximation do it?
• Can we do a fast match?
Fighting Bottlenecks

- Is there a faster algorithm?
- Would an approximation do it?
- Can we do a fast match?
- Can we parallelize?
Fighting Bottlenecks

- Is there a faster algorithm?
- Would an approximation do it?
- Can we do a fast match?
- Can we parallelize?
- Can we parallelize using specialized hardware?
Example: Iterative Optimization of ICSI Diarization System
Example: Iterative Optimization of ICSI Diarization System

- General Problem:
Example: Iterative Optimization of ICSI Diarization System

- General Problem:
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).
Example: Iterative Optimization of ICSI Diarization System

- General Problem:
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- Current Technical Issue:
Example: Iterative Optimization of ICSI Diarization System

- General Problem:
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- Current Technical Issue:
  - CPU parallelism limited and coarse but easier to implement
Example: Iterative Optimization of ICSI Diarization System

- **General Problem:**
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- **Current Technical Issue:**
  - CPU parallelism limited and coarse but easier to implement
  - GPU parallelism finer but harder to implement
Example: Iterative Optimization of ICSI Diarization System

- **General Problem:**
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- **Current Technical Issue:**
  - CPU parallelism limited and coarse but easier to implement
  - GPU parallelism finer but harder to implement

- **Current Solution:** Hybrid Approach:
Example: Iterative Optimization of ICSI Diarization System

- **General Problem:**
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- **Current Technical Issue:**
  - CPU parallelism limited and coarse but easier to implement
  - GPU parallelism finer but harder to implement

- **Current Solution: Hybrid Approach:**
  - CPU parallelism per cluster
Example: Iterative Optimization of ICSI Diarization System

- **General Problem:**
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- **Current Technical Issue:**
  - CPU parallelism limited and coarse but easier to implement
  - GPU parallelism finer but harder to implement

- **Current Solution: Hybrid Approach:**
  - CPU parallelism per cluster
  - GPU parallelism per frame
Example: Iterative Optimization of ICSI Diarization System

- General Problem:
  - Rewriting from a base of highly optimized serial code (~10k lines for the core algorithm).

- Current Technical Issue:
  - CPU parallelism limited and coarse but easier to implement
  - GPU parallelism finer but harder to implement

- Current Solution: Hybrid Approach:
  - CPU parallelism per cluster
  - GPU parallelism per frame

Process: Bottleneck by Bottleneck...
Speaker Diarization – Overview

Processing Chain:

Audio Signal →

Dynamic Range Compression → Wiener Filtering → Beamforming →

- Short-Term Feature Extraction → MFCC → Speech/Non-Speech Detector
- Long-Term Feature Extraction → EM Clustering

- Prosodics (only speech) → Initial Segments
- MFCC (only speech) →

- Prosodics (only speech) → Diarization Engine
- Delay Features

"who spoke when"

Overall Structure: Pipe & Filter
Speaker Diarization – Overview

Processing Chain:

Audio Signal

- Dynamic Range Compression
- Wiener Filtering
- Beamforming

- Short-Term Feature Extraction
  - MFCC
  - Speech/Non-Speech Detector

- Long-Term Feature Extraction
  - Prosodics (only speech)

- EM Clustering
  - MFCC (only speech)
  - Prosodics (only speech)

- Initial Segments

- Segmentation
- Diarization Engine
- Clustering

"who spoke when"

Overall Structure: Pipe & Filter
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
 Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
Speaker Diarization: Core Algorithm

Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Re-segment and repeat until no more merging needed
Algorithm outline:

- Start with too many clusters (initialized randomly)
- Purify clusters by comparing and merging similar clusters
- Resegment and repeat until no more merging needed
CPU Parallelization I

Easiest: Different Streams = Different CPUs

Parallelization: Each Feature Stream Processed in a Different CPU Thread (MapReduce)
CPU Parallelization I

Easiest: Different Streams = Different CPUs

Parallelization: Each Feature Stream Processed in a Different CPU Thread (MapReduce)
CPU Parallelization II

New Bottleneck: Comparison of clusters in each iteration

Parallelization: Each pair of comparison to a different CPU thread (MapReduce with irregular data access)
New Bottleneck: Comparison of clusters in each iteration

Parallelization: Each pair of comparison to a different CPU thread (MapReduce with irregular data access)
Serial optimization more efficient than parallelization!
CPU Parallelization II

New Bottleneck: Training Models

Parallelization: Each Cluster Computed on a Different Core (MapReduce of dense linear algebra operations)
CPU Parallelization II

New Bottleneck: Training Models

Parallelization: Each Cluster Computed on a Different Core
(MapReduce of dense linear algebra operations)
Diarization – CPU Parallelization

Parallelized Portion of the Runtime: ~62%
New Bottleneck: Computation of Log-Likelihoods
(Initially 28% of runtime, now 60% of runtime)

Parallelization: Use CUDA for Frame-Level Parallelization.
Diarization – GPU Parallelization

New Bottleneck: Computation of Log-Likelihoods
(Initially 28% of runtime, now 60% of runtime)

Parallelization: Use CUDA for Frame-Level Parallelization.
Diarization – GPU Parallelization

Result: Total Speaker Diarization speed-up 4.8 -> 0.07xRT
Next Week (Project Meeting)

Jaeyoung Choi on MediaEval 2012
Next Week (Lecture)

• Mid-Term Exam!