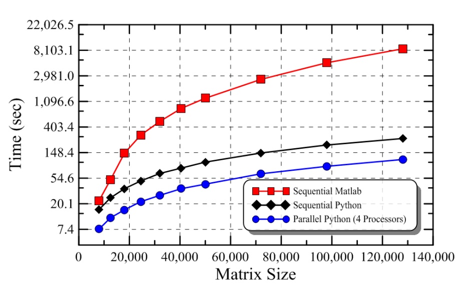
CS267 Final Project Presentations

May 5, 2016

Parallel Finite Element Analysis

Thanh Do, Parham Aghdasi, and Hussain AlSalem

1. Object-oriented finite element analysis toolbox
2. Python *multiprocessing* package
3. Parallel direct stiffness assembly
4. Parallel linear solver SuperLU

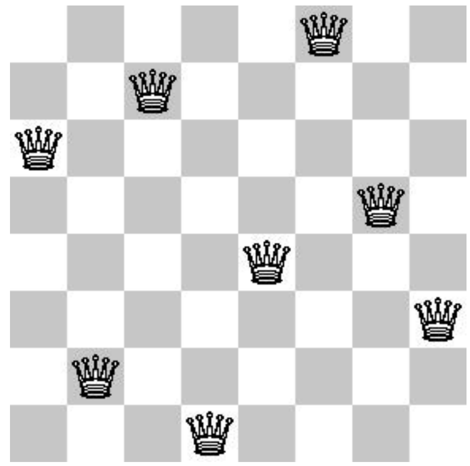



```
import numpy as np
import multiprocessing
    procs = 4
    ElemList = np.linspace(0,ne,procs+1,dtype='int')
    manager = multiprocessing.Manager()
return_dict = manager.dict()
    jobs = []

for ii in range(procs):
        process = multiprocessing.Process(target=Structure_2, args=
          ('stif', Model, ElemData, State, ElemList,ii, return_dict))
        jobs.append(process)
        process.start()

for proc in jobs:
        proc.join()
    State.Kf = return_dict[0].Kf

for ii in range(procs-1):
        State.Kf = State.Kf + return_dict[ii+1].Kf
```

Team 2: CHAITANYA ALURU, ACE HAIDREY, ROHIT MURALIDHARAN

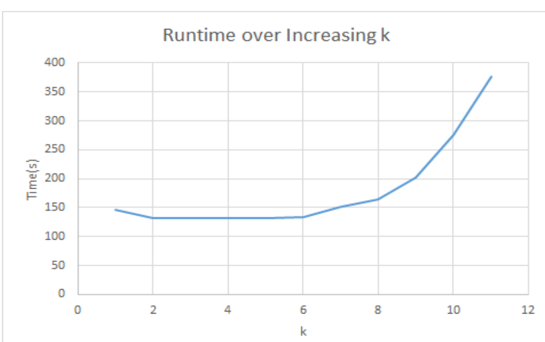
Parallel N-Queens

- Problem: Place *n* non-conflicting queens on an $n \times n$ board
- Serial Solution: Depth first search, place one queen at a time
- Parallelization: Can't just evenly divide search tree.
- Fix: Use Master Worker paradigm

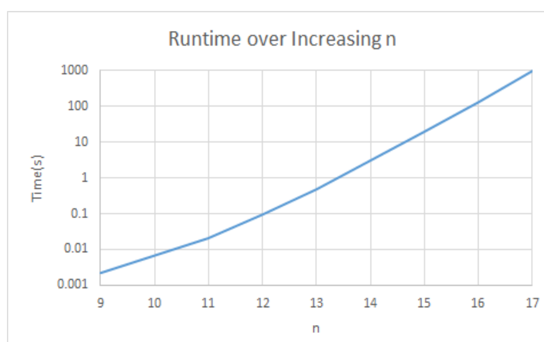


Team 2: CHAITANYA ALURU, ACE HAIDREY, ROHIT MURALIDHARAN

Runtime Improvements



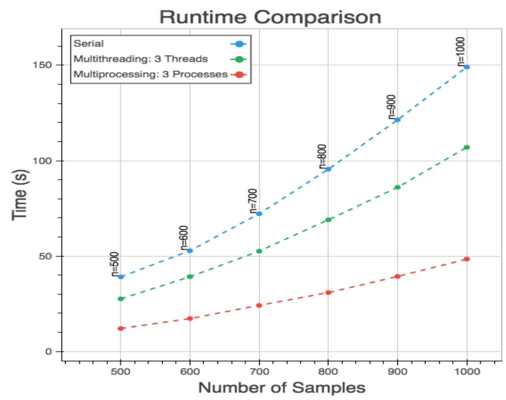
For n = 16, we saw consistent times for k-values of 2 through 6, before a steep increase as we went past 8.

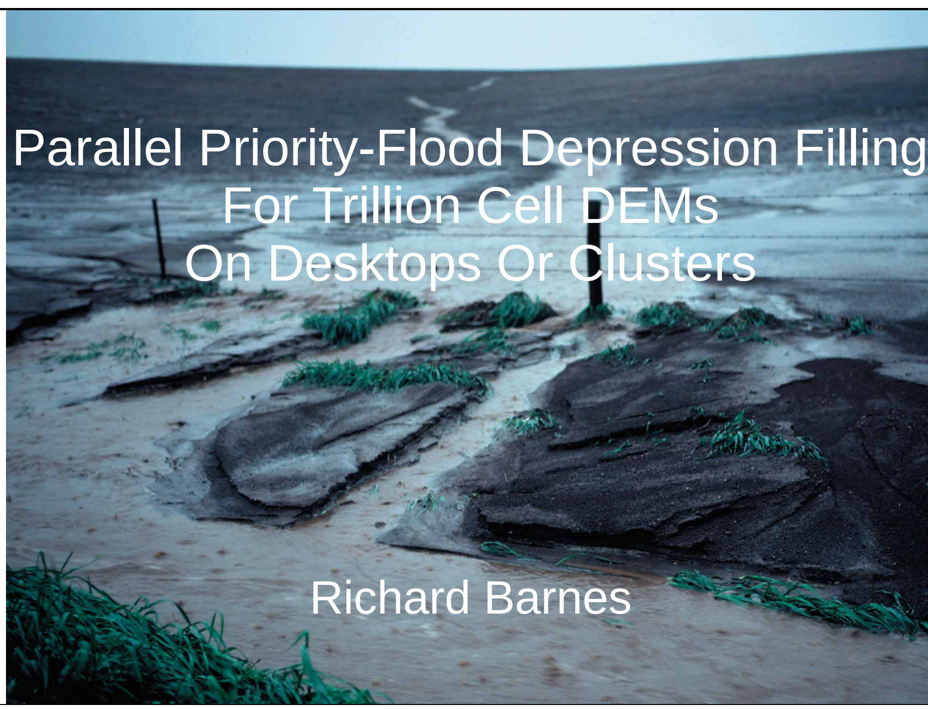


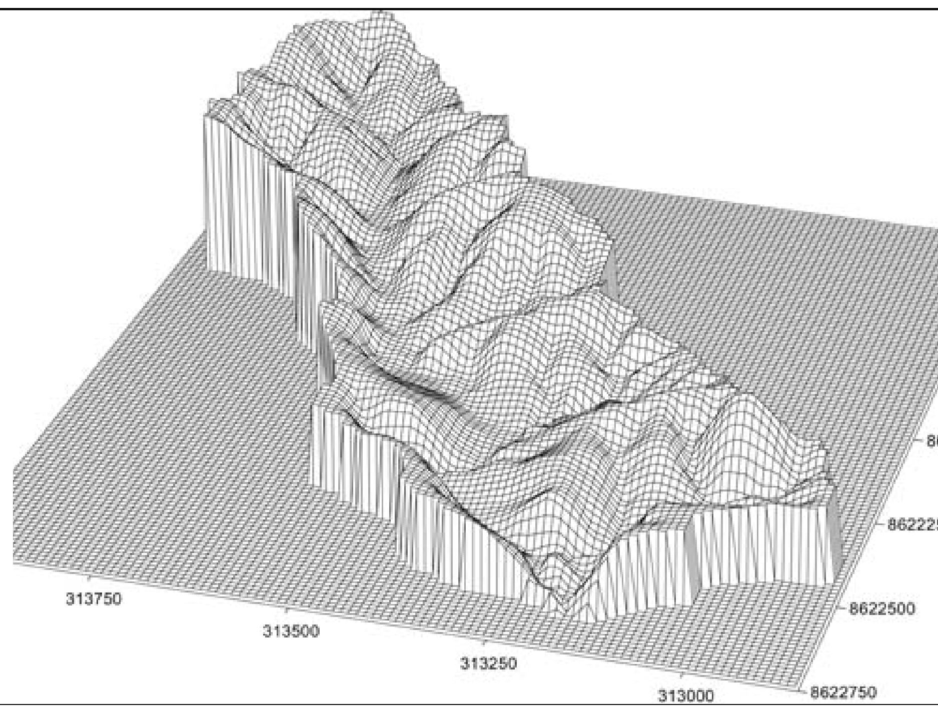
With k =4, we saw a steady exponential increase in runtime over increasing values of n.

Parallel Stable K-means

Emin Arakelian Fadi Kfoury Jason Poulos






313750
313500
313250
313000
8622750
8622500
862225
86


FIX
ME
© BNPS.CO.UK

Computers & Geosciences

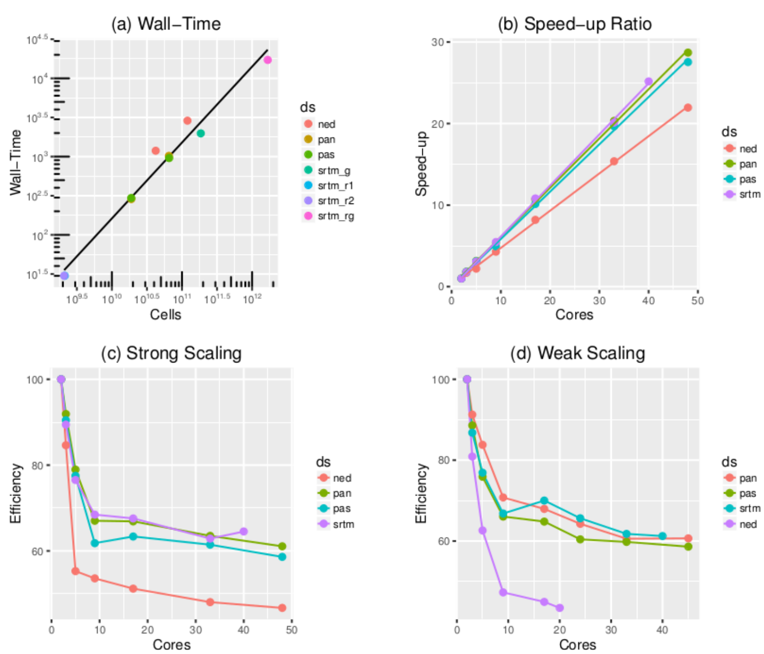
journal homepage: www.elsevier.com/locate/cageo

Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models

Richard Barnes [a,*], Clarence Lehman [b], David Mulla [c]

Source	Cells	Dimensions	Adjective	Time (min)	Min/Cell
Gomes et al. [10]	$3 \cdot 10^9$	50,000 x 50,000	huge	58	$1 \cdot 10^{-8}$
Do et al. [6]	$2 \cdot 10^9$	36,002 x 54,002	huge	21	$1 \cdot 10^{-8}$
Do et al. [7]	$2 \cdot 10^9$	36,002 x 54,002	huge	??	
Yıldırım et al. [27]	$2 \cdot 10^9$	45,056 x 49,152	large	??	
Arge et al. [2]	$1 \cdot 10^9$	33,454 x 31,866	massive	3720	$3 \cdot 10^{-6}$
Lindsay [14]	$9 \cdot 10^8$	37,201 x 25,201	massive	8.6	$1 \cdot 10^{-8}$
Tesfa et al. [22]	$6 \cdot 10^8$	24,856 x 24,000	large	20	$3 \cdot 10^{-8}$
Wallis et al. [24]	$4 \cdot 10^8$	14,949 x 27,174	large	8	$2 \cdot 10^{-8}$
Danner et al. [5]	$3 \cdot 10^8$	??	massive	445	$1 \cdot 10^{-6}$
Metz et al. [17, 18]	$2 \cdot 10^8$	??	massive	32	$6 \cdot 10^{-7}$

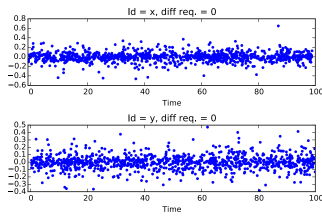
Source	Cells	Dimensions	Adjective	Time (min)	Min/Cell
This paper	$2 \cdot 10^{12}$	$\sim$1,291,715^2	*rather* large	291	$8 \cdot 10^{-9}$
Gomes et al. [10]	$3 \cdot 10^9$	50,000 x 50,000	huge	58	$1 \cdot 10^{-8}$
Do et al. [6]	$2 \cdot 10^9$	36,002 x 54,002	huge	21	$1 \cdot 10^{-8}$
Do et al. [7]	$2 \cdot 10^9$	36,002 x 54,002	huge	??	
Yıldırım et al. [27]	$2 \cdot 10^9$	45,056 x 49,152	large	??	
Arge et al. [2]	$1 \cdot 10^9$	33,454 x 31,866	massive	3720	$3 \cdot 10^{-6}$
Lindsay [14]	$9 \cdot 10^8$	37,201 x 25,201	massive	8.6	$1 \cdot 10^{-8}$
Tesfa et al. [22]	$6 \cdot 10^8$	24,856 x 24,000	large	20	$3 \cdot 10^{-8}$
Wallis et al. [24]	$4 \cdot 10^8$	14,949 x 27,174	large	8	$2 \cdot 10^{-8}$
Danner et al. [5]	$3 \cdot 10^8$	??	massive	445	$1 \cdot 10^{-6}$
Metz et al. [17, 18]	$2 \cdot 10^8$	??	massive	32	$6 \cdot 10^{-7}$



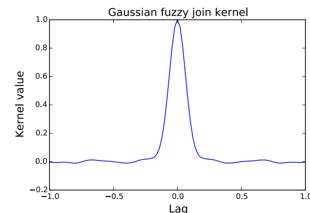
Temgine

Francois Belletti
francois.belletti@berkeley.edu

The scalable multivariate time series analysis engine



Irregular asynchronous data feeds



Implicit Gaussian fuzzy matching in frequency domain

$$SP_{XY}(f) = FT_X(f)\overline{FT_Y(f)} = \sum_{t \in \text{obs}(X)}^{t' \in \text{obs}(Y)} e^{-2\pi i f(t-t')} X_t Y_{t'}.$$

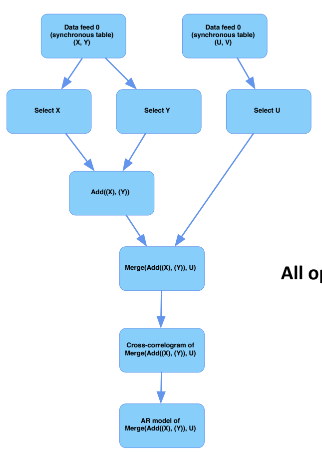
Leverage frequency-time domain duality to provide:

- Scalability
- Data agnostic use

$$IFT\left(SP_{XY}\right)(h) = \sum_{t \in \text{obs}(X)}^{t' \in \text{obs}(Y)} FT(F)(t - t' - h) X_t Y_{t'}.$$

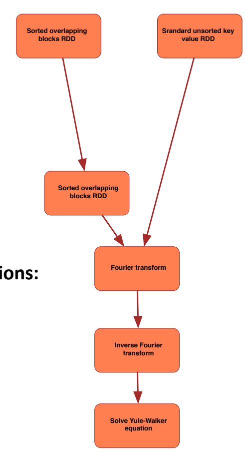
Graph of operations on processes

Logical execution flow in Temgine

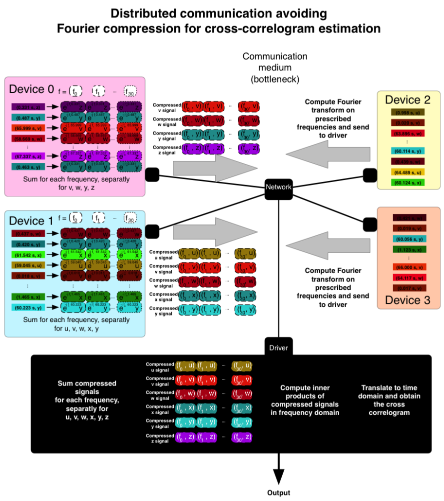


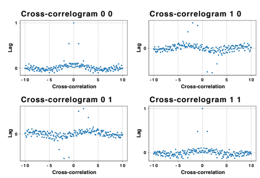
All operations have two definitions:
Time domain
Frequency domain

Physical execution flow in Temgine



Scalable linear model estimation

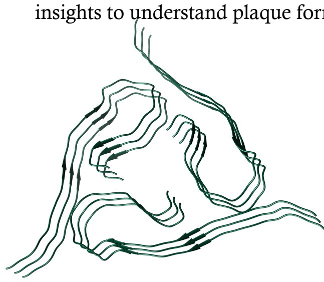




Inverting cross-correlation matrix yields a linear model (on master)

Parallelization of Coarse-Grained Molecular Dynamics

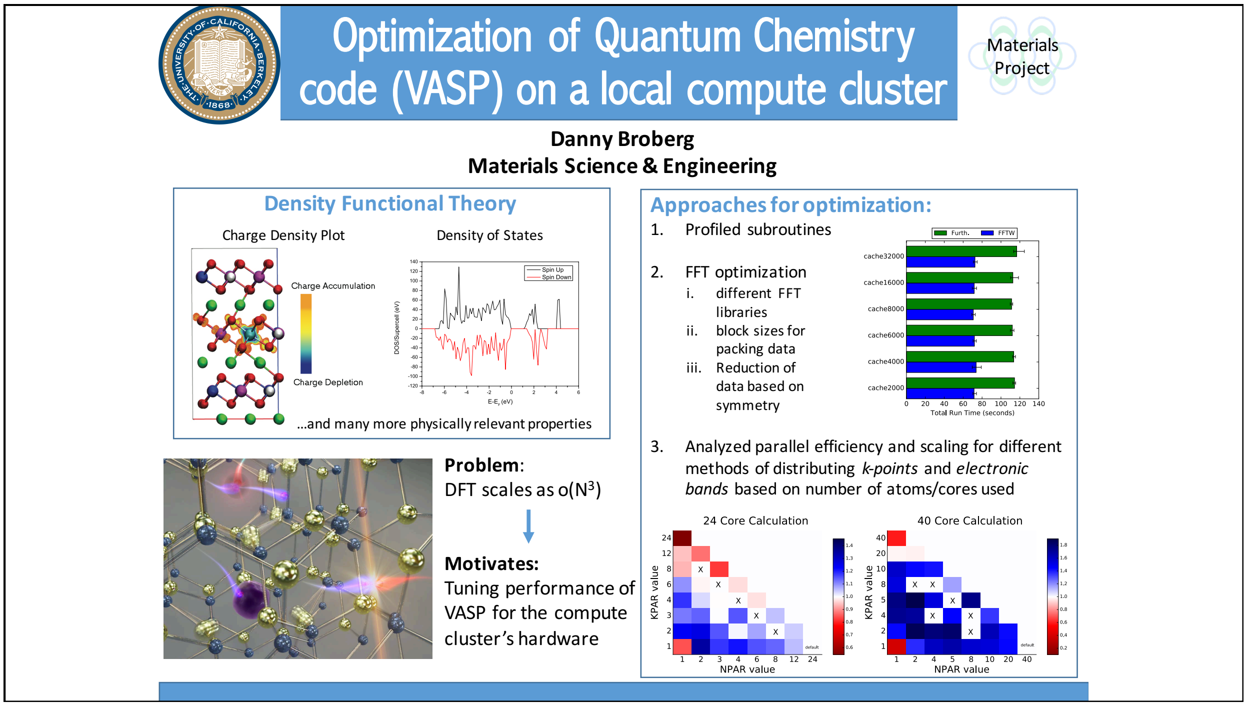
- Coarse-grained model to simulate amyloid protein aggregation.
- Implicit representation of solvent.
- Utilized to study protein thermodynamics.
- Faster code will enable us to reach longer timescales and hence better insights to understand plaque formation implicated in Alzheimer's disease.

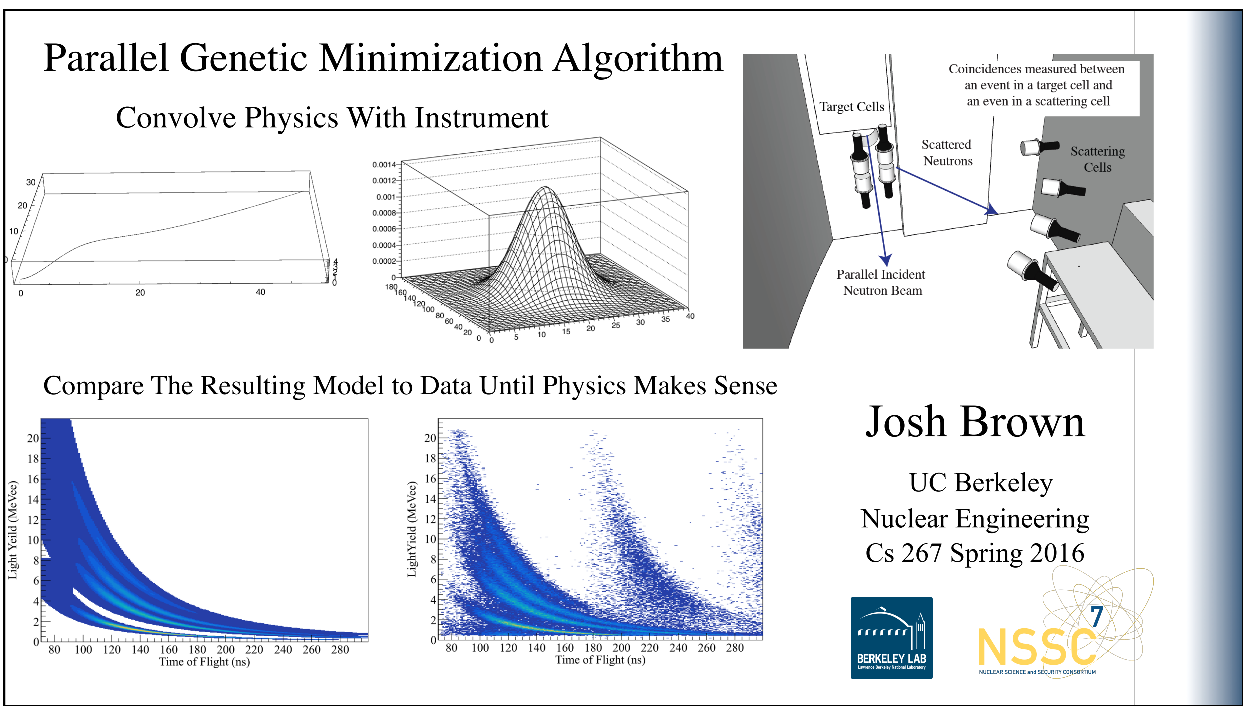


$$F = F_{bonded} + F_{non-bonded} + F_{random}$$

Parallelization Prospect

- Existing code is of $O(n^2)$
- Decomposition of the particles on a grid for calculating non-bonded forces
- Small system size ideal for OpenMP Parallelization


Optimization of Quantum Chemistry code (VASP) on a local compute cluster
Materials Project
Danny Broberg
Materials Science & Engineering
Density Functional Theory
Charge Density Plot
Density of States
Charge Accumulation
Charge Depletion
Spin Up
Spin Down
...and many more physically relevant properties
Problem:
DFT scales as o(N³)
Motivates:
Tuning performance of VASP for the compute cluster's hardware
Approaches for optimization:
1. Profiled subroutines
2. FFT optimization
i. different FFT libraries
ii. block sizes for packing data
iii. Reduction of data based on symmetry
Furth.
FFTW
cache32000
cache16000
cache8000
cache6000
cache4000
cache2000
Total Run Time (seconds)
3. Analyzed parallel efficiency and scaling for different methods of distributing k-points and electronic bands based on number of atoms/cores used
24 Core Calculation
40 Core Calculation
KPAR value
NPAR value


Parallel Genetic Minimization Algorithm
Convolve Physics With Instrument
Target Cells
Coincidences measured between an event in a target cell and an even in a scattering cell
Scattered Neutrons
Scattering Cells
Parallel Incident Neutron Beam
Compare The Resulting Model to Data Until Physics Makes Sense
Light Yeild (MeVee)
LightYield (MeVee)
Time of Flight (ns)
Josh Brown
UC Berkeley
Nuclear Engineering
Cs 267 Spring 2016
BERKELEY LAB
NSSC
NUCLEAR SCIENCE and SECURITY CONSORTIUM

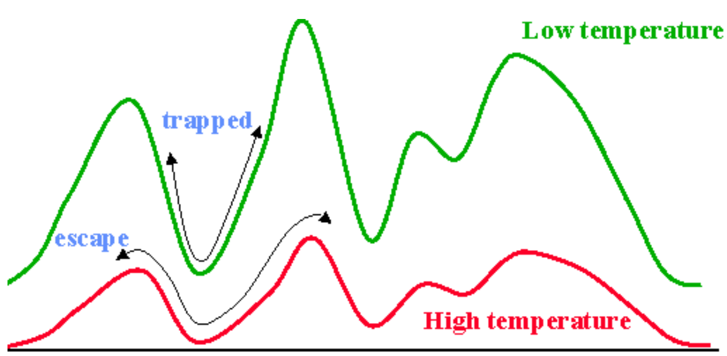
Parallel Tempering for reversible-jump Markov Chain Monte Carlo

Jane Yu and Jeffrey Chan

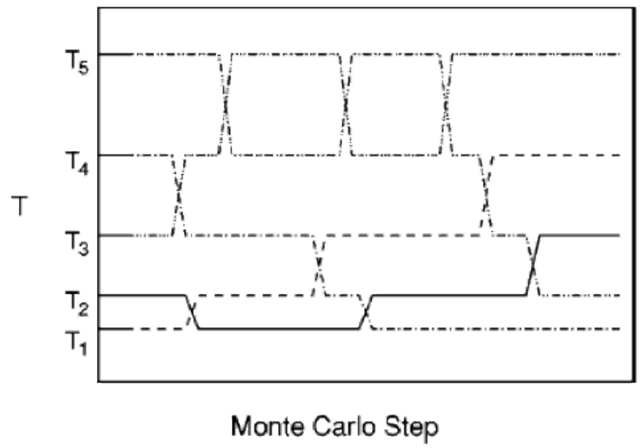
Computer Science Division, UC Berkeley

May 5, 2016

rjMCMC



Parallel Tempering Chains



Scaling Structure Learning via Parallel Computing

Xinlei Pan, Baiyu Chen

- DNA hybridization arrays measure the expression level of multiple genes simultaneously. We aim to find the underlying gene interactions using linear regression algorithm with regularization.
- Assume each node represents a gene, X, and the problem can be formulated as:

$$\min_{\beta_1,\ldots,\beta_p} \sum_{j=1}^{p} ||x_j - x'_{-j}\beta_j||_2^2 + \lambda \sum_{j=1}^{p} ||\beta_j||^q$$

- Estimating a lot of β takes a long time!
- The above problem can be solved using the generalized procedure:

$$\beta = \min_{\beta} ||Y - X\beta||_2^2 + \lambda||\beta||^q$$

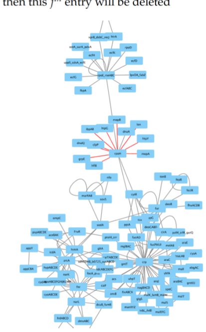
where $Y \in R_{N\times 1}$, $X \in R_{N\times p}$ and $\beta \in R_{p\times 1}$.It can be solved via the following algorithm according to [3]

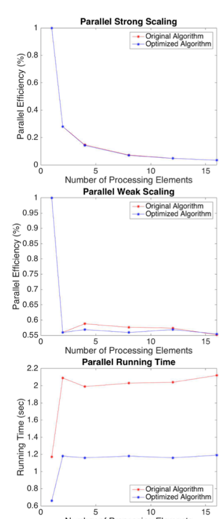
Initialization: Initialize $\beta = I$

Iteration: Calculate new β value:
$\beta^{(b)} = \{X_{-j}^T X + \Sigma_\lambda(\beta^{(b-1)})\}^{-1} X^T Y$,
where,
$\Sigma_\lambda(\beta^{b-1}) = \frac{\lambda q}{2} diag(|\beta_1^{(b-1)}|^{(q-2)}, ..., |\beta_p^{(b-1)}|^{(q-2)})$

Termination: Iterate until $||\beta^{(b)} - \beta^{(b-1)}|| < \eta$.

Variable Selection: During the iteration if $||\beta_j|| \leq eta$ then this j^{th} entry will be deleted



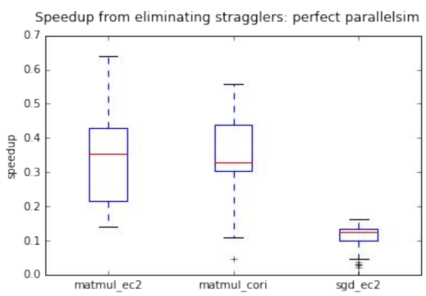


Blocked Time Analysis for Machine Learning Workloads on Distributed Computing System

Jiayuan Chen, Bill Kim, Jian Qiao

Methodology:
Blocked Time Analysis - end-to-end estimation of optimized job completion time

Workload: (Apache Spark cluster on EC2 & Cori)
Block-Matrix Multiplication
Logistic Regression with SGD



Results:
Infinitely fast network or disk I/O only gives limited amount of speedup
Task stragglers and CPU overhead are the bottleneck

Percent Reduction in Job Completion Time:

	No Disc	No Network	No Stragglers
EC2: BMM	4.5%	4.2%	32.7%
EC2: SGD	0%	0%	11.2%
Cori: BMM	2.8%	N/A	35.8%
Cori: SGD	TBD	N/A	TBD

Parallelization of PaSR: A Partially Stirred Reactor Model

Byung Gon Song, Jim Oreluk, Yulin Chen

Why does a PaSR matter?

- Between two idealized reactors. Perfect reactant mixture (PSR) & no reactant mixture (PFR)
- Offers alternate testbed for the evaluation of certain regimes in turbulent combustion
  - Simulations conducted with full chemical kinetics!
- Directly relevant to particle-tracking pdf for transport methods in multi-dimensional flow

Computational Pattern

"Monte Carlo Methods: A Computational Pattern for our Pattern Language" by Kurt Keutzer

- Numerical-centric perspective, task-centric perspective, data-centric perspective
  1) Structural Patterns : MapReduce
  2) Computational Patterns: Dense Linear Algebra

MapReduce is a programming model which is conceptually similar to Message Passing Interface having reduce and scatter operations. MPI was more suitable for our program because of following disadvantages of MapReduce:

1) Programming model is very restrictive
2) Cluster management is hard

```
for time = 1:end time
        for particle = 1:number of particles
                        computation on particles
        end
        reduction
end
output
```

Simplified model of the FORTRAN program

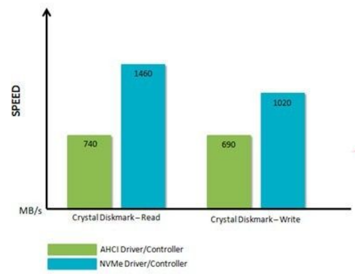
Results

- 28x speed-up on 46 processors for non-premixed methane
- More complex chemical mechanisms are no longer intractable and can be efficiently investigated

Parallel Algorithms on NVME

Richard Chiou, Harsha Simhadri

- Nonvolatile memories (e.g. Flash) are replacing DRAM.
- Writes are much more expensive than reads for nonvolatile memory - we can optimize algorithms accordingly.
- How does algorithm performance on a single node with NVM compare to that of a cluster with DRAM?
- We experimented on node F15 of LBNL's Firebox cluster.

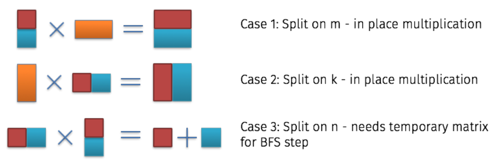


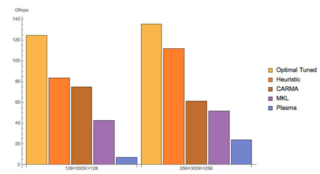
- BEN CORBETT
- LUCAS SERVEN
- RUNDONG TIAN

Space-Bounded Recursive PDF Scheduling

ASPIRE UC Berkeley
David Dinh
in collaboration with
Harsha Simhadri

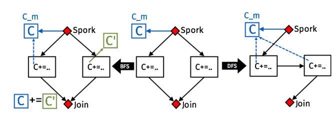
Tradeoff between **space** and **parallelism** in many nested parallel algorithms. For example, consider inner product-like algorithms. In tall-skinny matrix multiply (below), case 3 exhibits tradeoff. Making the wrong choices on tradeoffs can significantly degrade performance - order-of-magnitude difference between tuned tradeoffs and MKL!





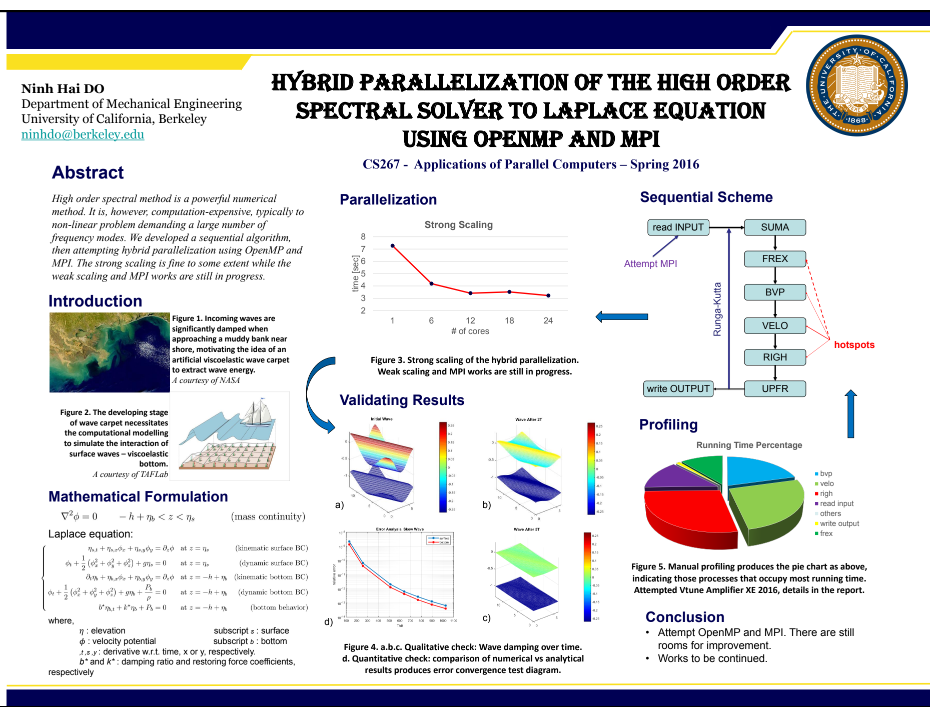
Interface based on Cilk Plus hyperobjects.

Restrict dynamic allocation only happens via hyperobjects in order to make it tractable for the the runtime to make decisions. Maintain invariants from space-bounded schedulers that guarantee optimality for cache and running time while using parallel depth-first ordering to achieve good space bounds.

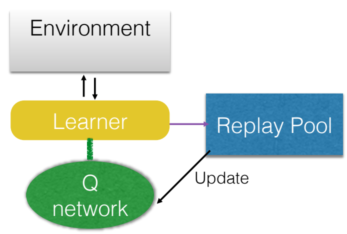


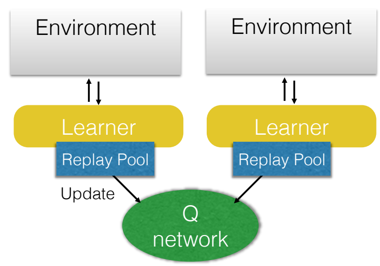
Implementation notes: being built using existing task scheduling framework on C++/pthreads.

Main tasks are to implement a locking doubly-linked list to store the tasks in depth-first order, and a way to handle online BFS/DFS decision-making.



Asynchronous Deep Deterministic Policy Gradient

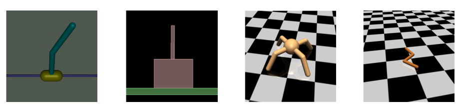




Minimize Loss:

$$L(\theta^Q) = \mathbb{E}_{s,a,s',r}[(Q(s,a|\theta^Q) - y(s,a,s'|\theta^Q))^2]$$

$$y(s,a,s'|\theta^Q) = r(s,a) + \gamma Q(s', \mu(s'|\theta^\mu)|\theta^Q)$$



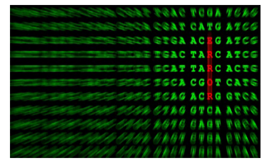
Results

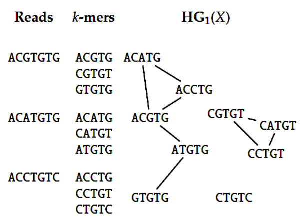
- Challenging to perform well under parallelization than other standard methods of RLs
- Good performance:
  - Sensitive to hyperparameters
  - Important to use replay pool

Distributed Memory Parallelization of Read Error Correction BayesHammer Algorithm

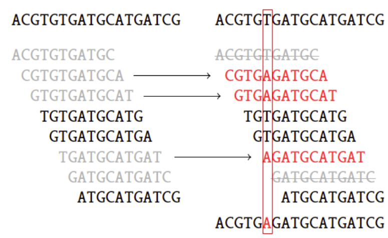
Sayna Ebrahimi

1. Constructing Hamming graph
2. Bayesian sub-clustering to find center of each k-mer's sub-cluster
3. Filter sub-cluster centers to form a set of solid k-mers.
4. Graph traversal and counting the majority vote of solid k-mers





Hamming graph* of distance 1



Error correction of a contig*

* Nikolenko, Sergey I., Anton I. Korobeynikov, and Max A. Alekseyev. "BayesHammer: Bayesian clustering for error correction in single-cell sequencing." *BMC genomics* 14.1 (2013): 1.

Multistate Protein Design Optimization

Motivation:

- Multistate protein models are better at capturing the natural flexibility of proteins
- The large number of hyperparameters significantly affect model performance

Goal: Parallelize a cuckoo search optimization algorithm (biased random walk) to search through the hyperparameter space.

Computational Patterns: Map Reduce and some n-body

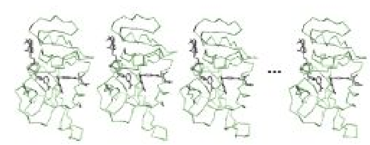
Structural Pattern: Iterator

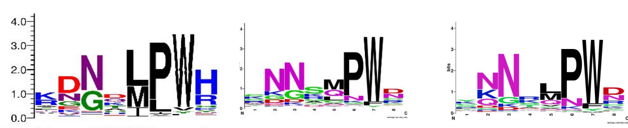
Results:

- 3 orders of magnitude increase in speed
- Can now run locally instead of on cluster

Key Parallelization Details:

- Armadillo Library for matrix computations (includes LAPACK and BLAS)
- Load balancing and synchronization
- Scale for population size, scale for number of iterations

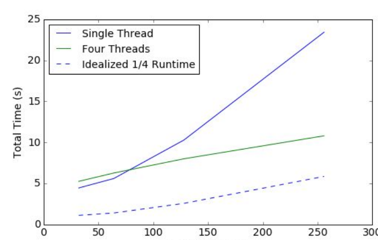




Bacterial natural DHFR M20 loop profile

Optimization from python implementation (>24 hours)

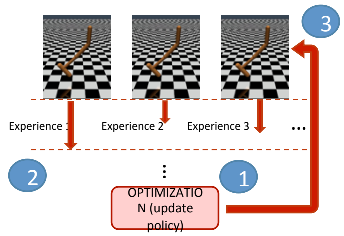
Optimization from C++ parallel implementation (under 5 minutes)



Tianjiao Zhang, Jenelle Feather
CS267 Final Project, 2016

Parallelizing Reinforcement Learning

Some parallelism is easy in RL…

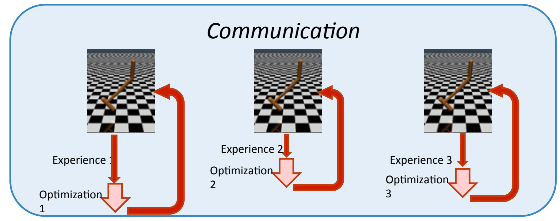


… But bottlenecks prevent high performance & scaling

A. Optimization and experience processing are done centrally, (often by a **single thread**), requiring significant data transfer
B. Often **poor load-balancing** (e.g., experiment length varies)
C. Require all processors to **equalize parameters** before continuing

Synchronization cost

Our proposed setup



- Each process performs some **optimization locally**
- **Share** only the **update** (or current parameters)
- Can run **asynchronously and lock-free**
- **Key question:** how to manage communication?

MPI

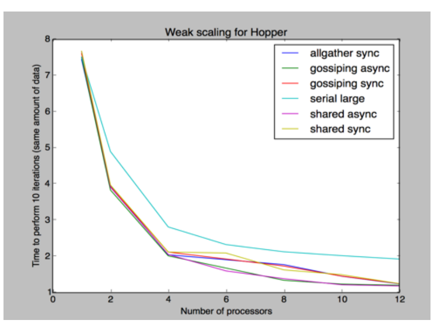
Allows **divergent policies**:
- May improve exploration
- Delicate for optimization

Shared memory

Less communication:
- More up-to-date params
- Might scale worse

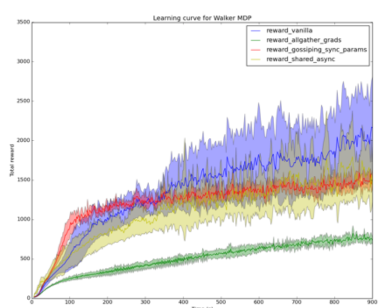
Our algorithms scale better, but performance impact is unclear

Scaling



All methods we propose **scale better**. Notice scaling worsens after 6 processors due to hyperthreading on the machine on which we tested

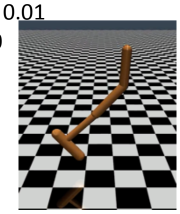
Algorithm performance



Our learning is often **faster in early iterations**, but we are **overtaken** by the classical method (is hyper-parameter tuning to blame?)

Experiment details

Hardware
Processor: Intel Core i7-5280K CPU @3.30GHz x12
Graphics: GeForce GTX TITAN X
Memory: 32GB
Problem instance
Test MDP: MuJoCo Hopper
Total steps per iter: 36.000
Time horizon: 1000
Learning rate: 0.01
Iterations: 500



Efficient Synchronous Stochastic Gradient Descent

reducing communication cost in Synchronous SGD

Previous Method

```
Input: T, B, init_params
Output: final_params
for t = 1 to T
    load_batch(B);
    calc_gradient();
    sync();
    update_params();
end
```

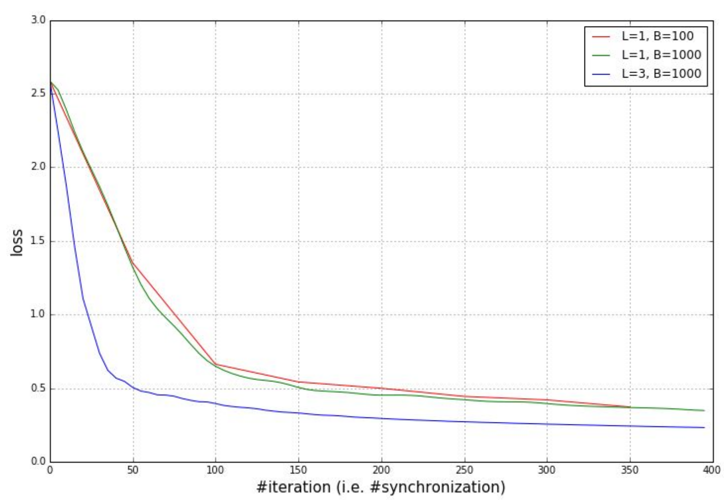
Too much communication cost

Our Method

```
Input: T, L, B, init_params
Output: final_params
for t = 1 to T/L
    for l = 1 to L
        load_batch(B);
        calc_gradient();
        update_params();
    end
    sync();
end
```

Reduce loss with less synchronization

Experiment on MNIST dataset.
Ours (blue) achieves the same loss with significantly less synchronization.



MPI Parallelization of AMRStencil Framework

CS267 Final Project

Chris L. Gebhart
Jingyi Wang

Berkeley
UNIVERSITY OF CALIFORNIA

AMRStencil

AMRStencil:
- Framework for a domain specific programming language

Designed to facilitate:
- Calculations on unions of rectangles
- Stencil calculations
- Domain refinement

Only functional in serial, now moving to parallel!

Berkeley
UNIVERSITY OF CALIFORNIA

Progress

- Implement a non-trivial serial code for testing 4th order finite volume solver for Shallow Water
- Rewrite key structures for MPI capability
- One domain patch per processor
- No adaptivity (spatial or temporal)
- Simple rectangular domain

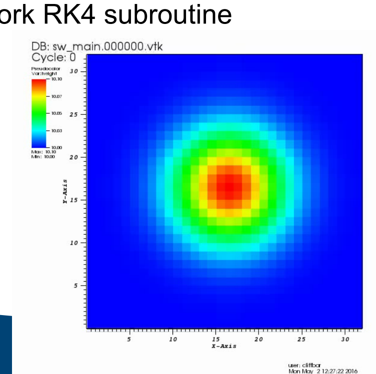
Berkeley
UNIVERSITY OF CALIFORNIA

Results

Serial Code:
- Correctly solves simplified version of Shallow Water Equations

Parallelization:
- A 40.3% percent speedup on 4 processors with respect to serial
  Huge opportunity for speedup: rework RK4 subroutine

Future work:
- Optimize code for performance
- Implement multithreading via OpenMP
- Implement Adaptive Mesh Refinement



Berkeley
UNIVERSITY OF CALIFORNIA

Atmospheric Simulations for Next Generation CMB Experiments

N. Goeckner-Wald, J. Savarit, R. Keskitalo, J. Borrill

- The next generation of CMB experiments will face significant data volume O(10 pB) and computation challenges, including the need for O(10,000) Monte Carlo realizations to establish systematic errors)
- The atmosphere is a significant contaminant in data
- Current modeling paradigms scale quadratically in detector number (data volume) due to large dense matrix eigenvalue decompositions
- Needed: A scalable, parallel Monte Carlo noise generation code

POLARBEAR CMB Experiment

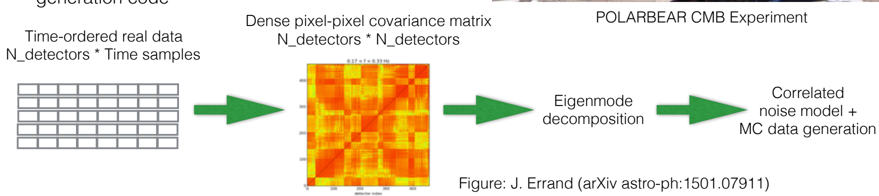
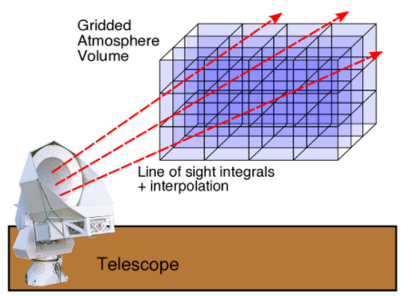


Figure: J. Errand (arXiv astro-ph:1501.07911)

Significant computational and physical challenges

- Accurate noise models contain many physical and computational challenges
- Physical: Atmospheric dynamics on relevant scales is not constrained by existing data
- Computational: Simulating atmospheric spectrum over very large atmospheric volumes requires an efficient 3D FFT operation
- Computational: Effectively exploit shared-memory parallelism to quickly perform many line of sight integrals for each time step over a simulated atmosphere
- Physical: What level or realism in noise modeling is sufficient to capture the pixel-pixel covariances seen in real data?
- Computational: The need to run O(10,000) realizations of a single experiment containing O(10,000) single observations



Physically Motivated Atmospheric model → Line of sight integrals using telescope pointing, interpolate over focal plane

Based on work by J. Errand
(arXiv astro-ph:1501.07911)

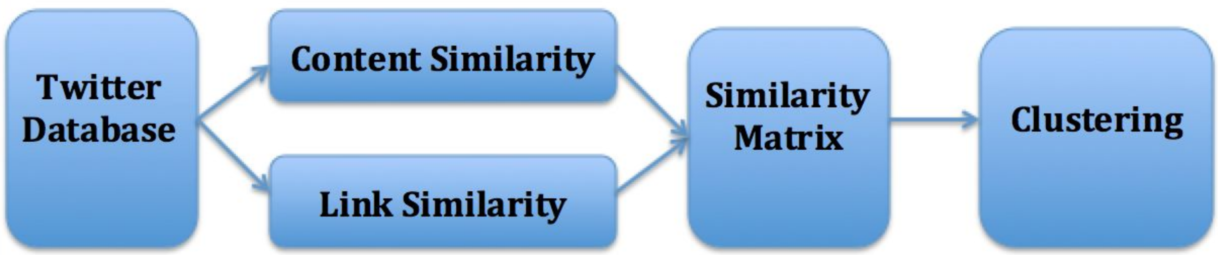
Shot Boundary Detection
w/ PyCuda

Alex Hall

Parallel Computing for Community Detection

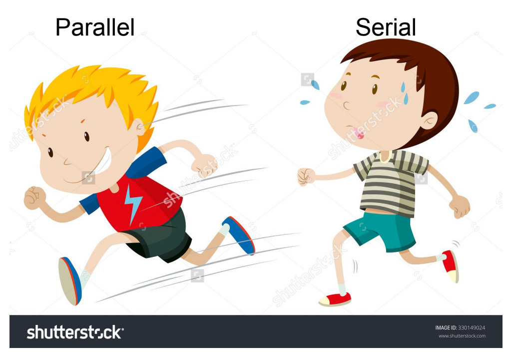
By: Goutam Murlidhar, Alper Vural,
Alagu Sanjana Haribhaskaran,

System Architecture



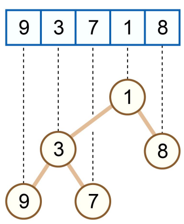
What This Means

- Processes Made Easier
  - Analyzing large graphs like facebook, twitter networks
  - Clustering biological data
  - Analyzing Power Grids

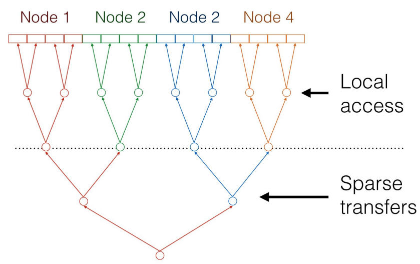


Parallelizing Cartesian Tree Construction on Multiple Nodes with PGAS

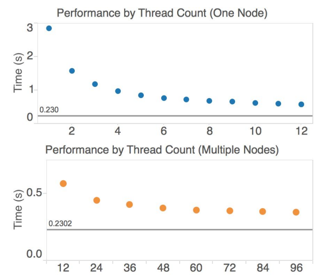
Andrew Head



Problem



Insight



Results

This work extends a parallel algorithm by Shun & Blelloch, 2014, *A Simple Parallel Cartesian Tree Algorithm*….
Cartesian tree photo modified from source at: https://upload.wikimedia.org/wikipedia/commons/d/d5/Cartesian_tree.svg

Twitter Language Specifier on Spark with GPU

Qifan Pu

James Jia

Qijing Huang

1

Background – Twitter App, Spark and GPU

- **Twitter Language Specifier App**
  1. Collect a Dataset of Tweets
  2. Train a Kmeans Model
  3. Apply the Kmeans model in real-time

Our Goals:

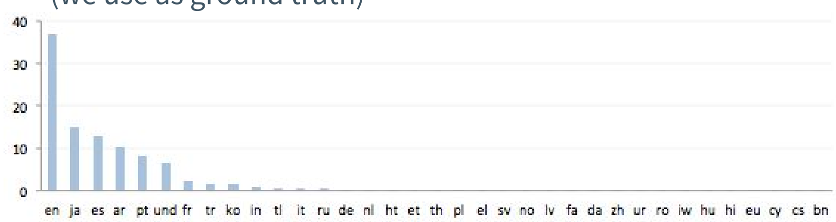
Accelerate step 2 & 3 using GPUs

- **Spark**
  - A fast and general big data processing engine with streaming interface
- **GPU**
  - Good at running massive computational intensive parallel jobs

2

Collect/ Twitter Data

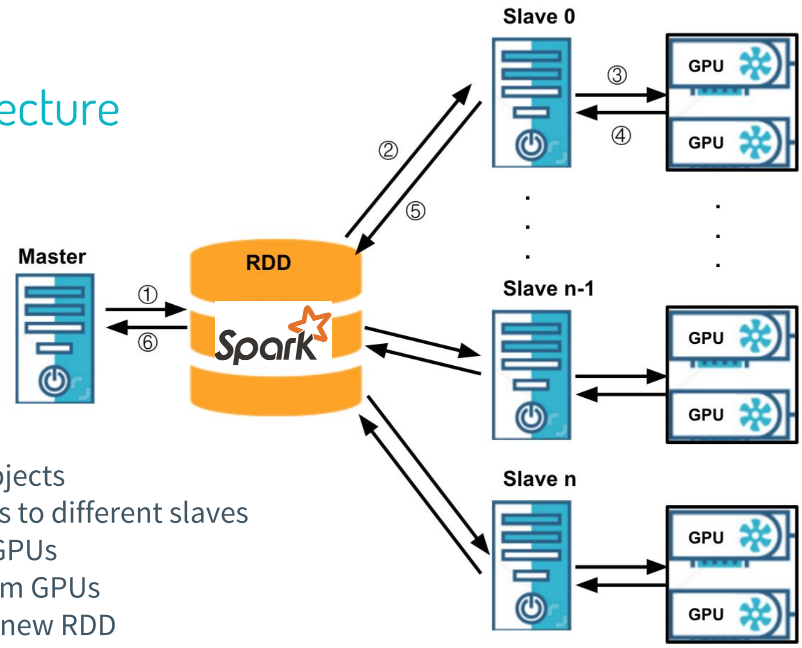
- Training:
  - Collect twitter data with varied sample sizes
  - Each tweet comes with detected-language information (we use as ground truth)



- Prediction:
  - Use twitter streaming API to get live, real-time tweets
  - Approx. 60 tweets per second, predict on various batch sizes

3

Overall Architecture



1. Create RDD for objects
2. Map the partitions to different slaves
3. Transfer data to GPUs
4. Return results from GPUs
5. Output results to new RDD
6. Collects result from the RDD

- ILIOPOULOS,FOTIS MANURANGSI,PASIN WONG,SAM

Hogwild!

- **Asynchronous** framework for parallelizing algorithms for **convex** and **sparse** problems
  - E.g. stochastic gradient descent
- Question: is convexity *necessary*?
- Constraint satisfaction problems (CSP) are **highly non-convex**
  - CSP ≈ satisfy many clauses; clause ≈ logical formula
- Does Hogwild still work for CSP?

Parallel CSP

Heuristic for solving CSP

1. Start with an assignment
2. Fix an unsatisfied clause by resampling an assignment *for its variables* randomly
3. Repeat!

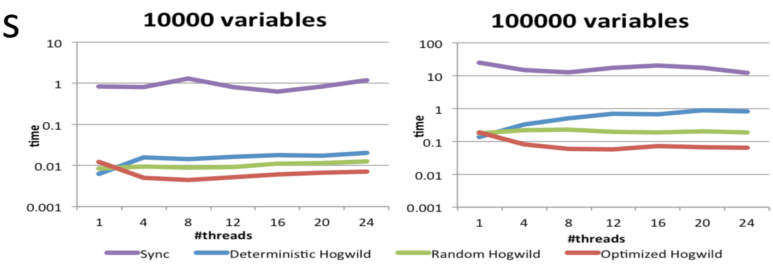
Provably works for sparse instances (Lovasz Local Lemma)

Can we parallelize in Hogwild fashion?

Parallel CSP with Hogwild!

Idea: each thread fixes unsatisfied clauses in *parallel*

Sync: provably works but need **locks**



Hogwild!: **asynchronous**; 2 threads may try to change same variable

Deterministic: fix *first* violated clause
Random: fix a *random* violated clause
Optimized: **recursively** fix *random* violated clauses

Hogwild! works – **massive scaling; non-convex OK**

Group 29

GPU NUFFT (Non-uniform FFT)

Teresa Ou, Wenwen Jiang, and Frank Ong

Motivations

- Iterative MRI and tomographic imaging reconstruction:

  NUFFT (Non-uniform FFT) is computational bottleneck

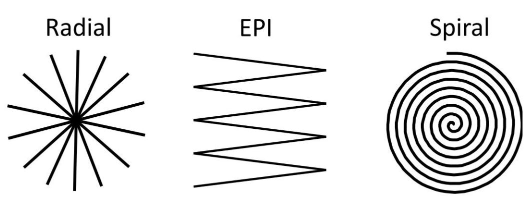
- Current toolboxes:

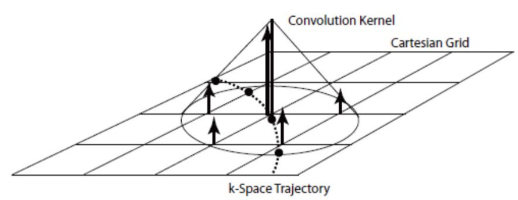
  Most do not exploit the parallel computing architectures

Goal

Develop optimized NUFFT for GPU

Methods: GPU NUFFT (Non-Uniform FFT)

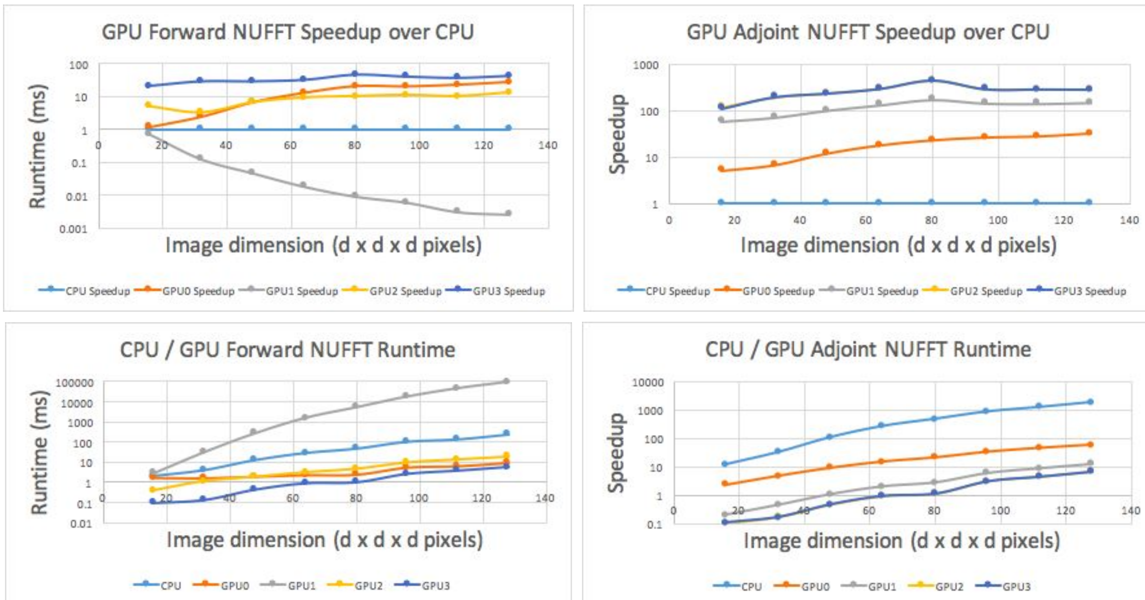




⇒ Interpolate onto oversampled Cartesian grid
⇒ Oversampled FFT
⇒ Deapodize (pointwise multiplication)

- Coordination of grid updates
- Forward and adjoint 3D NUFFT
- Optimizations: sparse matrix and fftshift

Performance



Benchmarking Communication-Efficient Distributed Optimization Algorithms on Spark

Chi Jin

For large scale datasets and ML tasks run on cluster,

$$\min P(w), \qquad P(w) := \frac{1}{n}\sum_{i=1}^{n}\psi_i(w).$$

what is the best communication-efficient algorithm.

Algorithm 4: Template of DMB algorithm for stochastic optimization.

$r \leftarrow \lfloor \frac{m}{b} \rfloor$;
for $j = 1, 2, \ldots, r$ **do**
  reset $\hat{g}_j = 0$;
  for $s = 1, \ldots, b/k$ **do**
    receive input z_s sampled i.i.d. from unknown distribution;
    calculate $g_s = \nabla_w f(w_j, z_s)$;
    calculate $\hat{g}_j \leftarrow \hat{g}_j + g_i$;
  end
  start distributed vector sum to compute the sum of $\hat{g}_j$ across all nodes;
  finish distributed vector sum and compute average gradient $\bar{g}_j$;
  set $(w_{j+1}, a_{j+1}) = \phi(a_j, \bar{g}_j, j)$;
end
Output: $\frac{1}{r}\sum_{j=1}^{r} w_j$

Mini-batch SGD?

LBFGS, Splash, CoCoA/CoCoA+

Results:

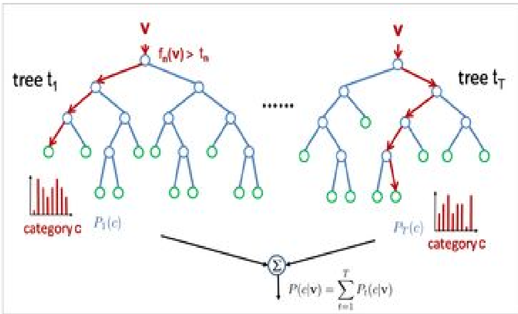
1. The description of essential ideas and applicability of above algorithms.
2. A comparison of their theoretical guarantees.
3. A comparison of their performance on Cori on benchmark datasets.

Parallel Random Forests

Nikhil Narayen, Aleks Kamko

Serial Random Forests

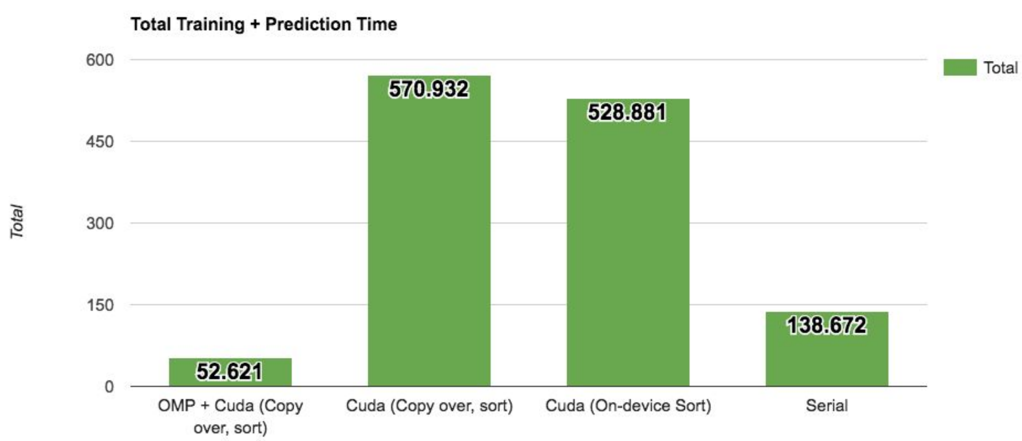
- Ensemble method for learning and classification
- Main motivation is to train many 'dumb learners' to address overfitting
- MNIST data set
  - 60,000 images for all 10 digits



Parallel Random Forests

- Many areas for parallelization for Random Forests
  - Simultaneous tree training
  - Breadth-first node training
  - Feature splitting
- Our optimizations
  - Using Thrust library to sort feature data in parallel
  - Using OpenMP to parallelize training of individual trees and nodes
  - Parallelizing finding the best split amongst a group of candidate features

Initial Results, 10K samples



Initial Results, 10K samples

Initial Results, 10K samples

- The overhead of copying data to the GPU is too high, as seen in the performance drop in our Copy-Over, Sort CUDA implementation
- The overhead for organizing data with a single GPU thread is also too high, as seen in the performance drop in our On-Device Sort CUDA implementation
    - Although, this implementation performs slightly faster than copy-over
- Using OMP to sort multiple arrays *simultaneously* on the GPU overcomes the overhead of copying data into the GPU

Hypotheses for Future Work

We expected GPU sorting to bring down forest training time, but this wasn't the case in our initial experiments! We think that we observed this for one or more of the following reasons:

- There is some cutoff number of elements below which it isn't worth sorting with CUDA
- Processing sorted data asynchronously could shave off more time
- Using multiple GPU threads to organize feature data could potentially speed up the On-Device sort implementation

We plan to experiment with these issues in the next few days.

ZIPG: SERVING QUERIES ON COMPRESSED GRAPHS

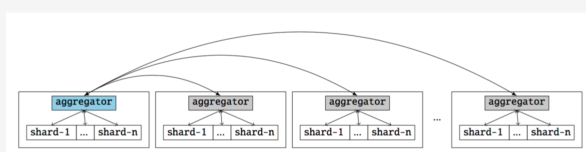
Anurag Khandelwal

- Distributed Graph Stores
  - **Social networks:** FB, Twitter, LinkedIn
- Challenges:
  - Graphs are **huge**
    - Facebook: ~10^9 nodes, ~10^{12} edges, with rich attributes
  - Graph queries are **complex**
    - E.g.: "Friends of my friends who like video games"
  - **Strict** performance requirements
    - Low latency, high throughput
- Our Approach:
  - **Compression** helps **caching**
  - Decompression is **expensive**
    - Execute queries **directly on compressed representation**
  - Graph layout that is
    - **flexible**, **amenable to compression** & can **scale**.

Adjacency List, augmented with properties

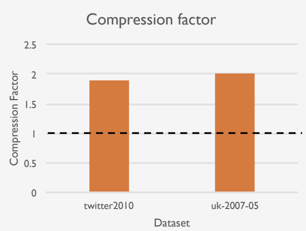
Source	Edge Type	Timestamps	Destinations	Properties
S_1	0	$T_1, ..., T_n$	$D_1, \ldots, D_n$	$P_1, \ldots, P_n$
S_2	1	$T_1, ..., T_n$	$D_1, \ldots, D_n$	$P_1, \ldots, P_n$
S_3	1	$T_1, ..., T_n$	$D_1, \ldots, D_n$	$P_1, \ldots, P_n$

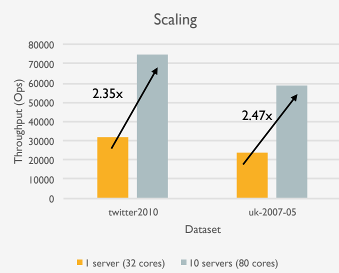
1. Partition by source
2. Compress using Succinct [NSDI'15]; supports:
   - decompression at random offsets
   - search for arbitrary patterns w/o decompression
3. Distribute

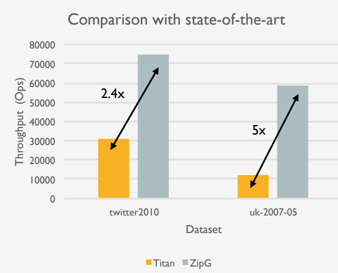


ZIPG: SERVING QUERIES ON COMPRESSED GRAPHS

- Datasets:
  - twitter-2010 (social graph, 41 M nodes, 1.4 B edges, 250GB raw data)
  - uk-2007-05 (web graph, 105M nodes, 3.7 B edges, 636GB raw data)
- Facebook's TAO Workload (99.8% reads, .2% writes & updates)







Takeaway: ~2x compression, with close to linear performance scaling, and 2.4 - 5x higher throughput than state-of-the-art

Cache Friendly Shuffles for ML

Maximilian Lam Horia Mania Maxim Rabinovich

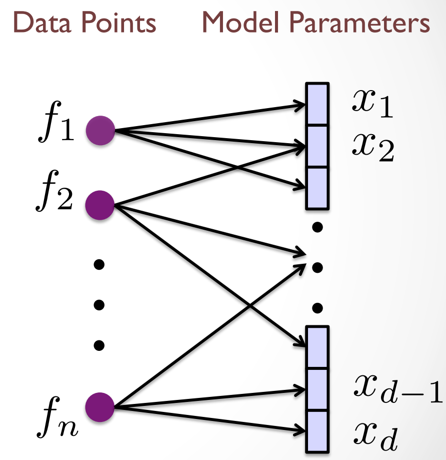
$$\min_{x \in \mathbb{R}^d} F(x) = \sum_{i=1}^{n} f_i(x)$$

loss for data point i

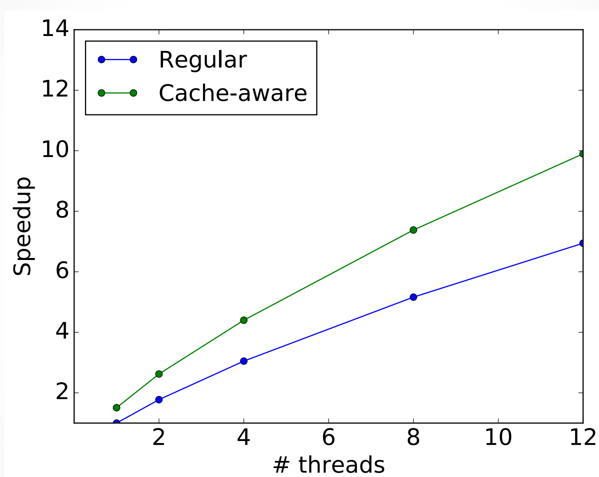
- **Goal:** Minimize sum of losses
- **Idea:**
  - Go over each data point, and locally optimize.
  - Multiple threads perform independent updates.

Access Pattern

- **Now:** Go over data in random order
- **Our approach:** choose orderings that improve cache locality



An experimental Result



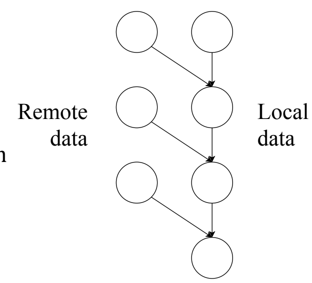
- Word2Vec: 20m data points, 200 features

Communication Minimization Under Approximate Correctness

Ke Li

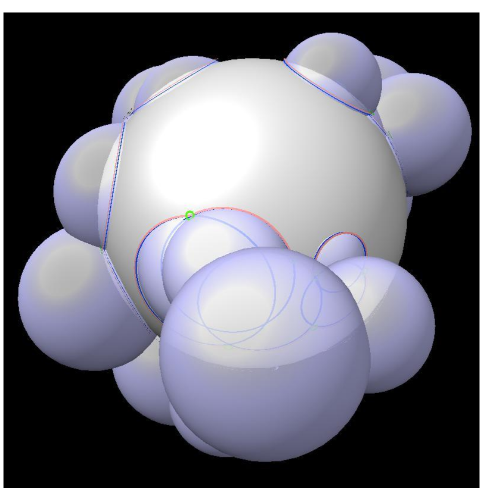
- Most existing work focuses on the *exact* setting; we consider the *approximate* setting.
- Given a parallel algorithm, we devise a way to determine which variables should be communicated.
- For variables that are not communicated, we use randomly sampled values.
- We show the quantity we should control is the sum of square roots of the *mutual information* between the variables and the output.

$X_1, \dots, X_n$: uncommunicated variables
Y: algorithm output, which is in $[l_1, l_2]$
If $\sum_{i=1}^{n} \sqrt{I(X_i; Y | X_{i+1}, \dots, X_n)} < \delta / \left(\sqrt{2}(l_2 - l_1)\right) - \sqrt{\log(2/\epsilon)}$,
then the output deviates by at most δ with probability of at least $1 - \epsilon$.



Problem Statement

Xingjie Pan
Yuhao Liu
Yanrong Li



GPU Implementation of SASA (Solvent Accessible Surface Area)
Calculate accessible surface on a union of spheres

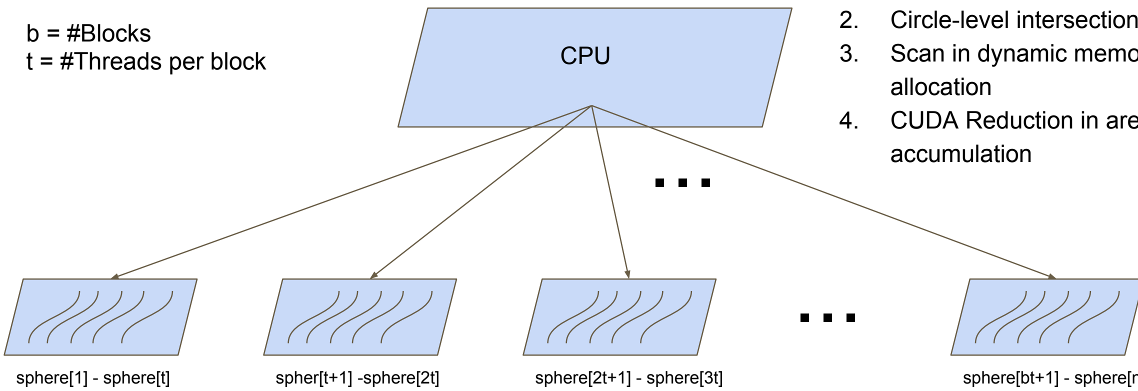
Applications

- **Molecular Dynamics**
- **Protein Structure Prediction**

Algorithms

Xingjie Pan
Yuhao Liu
Yanrong Li

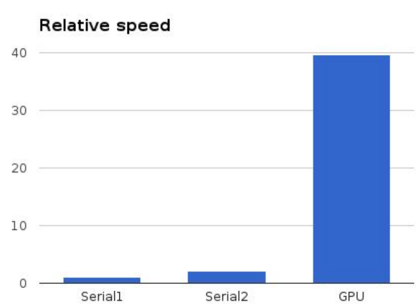
b = #Blocks
t = #Threads per block

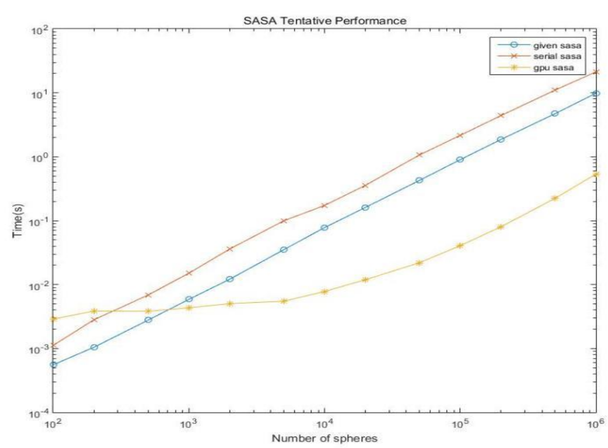


1. Sphere-level parallelism
2. Circle-level intersection
3. Scan in dynamic memory allocation
4. CUDA Reduction in area accumulation

Results

Xingjie Pan
Yuhao Liu
Yanrong Li



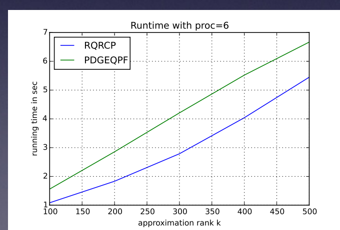


A parallel implementation of RQRCP

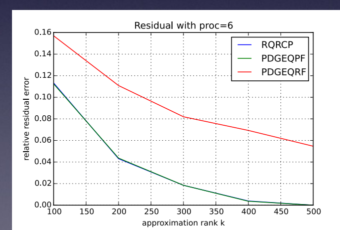
Jianwei Xiao & Ruochen Liang

- Problem: do partial QR factorization with pivoting to find low rank approximation
- Previous work: RQRCP based on OpenMP on shared memory machines
- Existing method: ScaLAPACK subroutine PDGEQPF
- Our work: RQRCP based on ScaLAPACK on distributed memory machines
- Result: less running time, similar approximation

Running time comparison



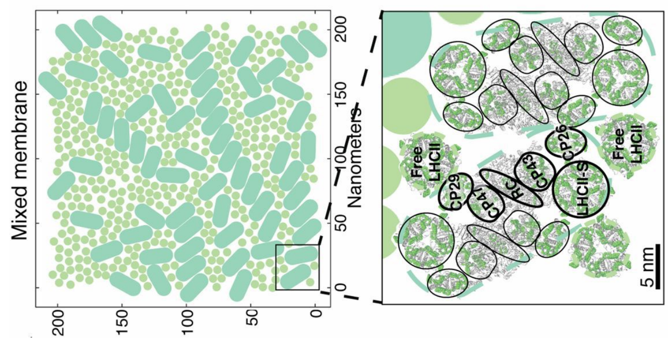
Relative residual error comparison



Modeling Energy Transfer Dynamics in Thylakoid Membranes: Parallel Matrix Exponentiation

Jonathan Morris
Marten Lohstroh

Photosynthesis: Energy Transfer on the Thylakoid Membrane



Mesoscopic Effects of Quantum Dynamics: Classical Rate Matrix

Interdomain Transfer Described by Generalized Förster Theory:

$$k_{M\leftarrow N} = \frac{|V_{M,N}|^2}{\hbar^2 \pi} \mathrm{Re} \int_0^\infty \mathrm{d}t A_M(t) F_N^*(t),$$

Intradomain Transfer Described by Modified Redfield Theory:

$$k_{M\leftarrow N} = 2\mathrm{Re} \int_0^\infty \mathrm{d}t A_M(t) F_N^*(t) V_{M,N}(t),$$

Dynamics is given by first order ODE:

$$\dot{P}(t) = \mathbf{K}P(t)$$

Formal solution:

$$\vec{P}(t) = e^{\mathbf{K}t}\vec{P}(0)$$

–

Reduced Algorithmic Complexity: Krylov Subspace Method

Naive Solution:

$$e^{\mathbf{K}t} = \mathbf{V}e^{-\lambda t}\mathbf{V}^{-1}$$

Krylov Subspace Projection:

$$\mathbf{A}\mathbf{V}_m = \mathbf{V}_{m+1}\mathbf{H}_m$$

Compute Solution on smaller subspace:

$$\vec{P(t)} = ||\vec{P}(0)||_2 \mathbf{V}_{m+1} e^{\mathbf{H}_m t}\vec{e}_1$$

Compute on the smaller subspace via scaling and squaring;
Padé Approximant Techniques.

$$e^{\mathbf{H}} = \left(e^{\mathbf{H}/2^j}\right)^{2^j} \quad e^{\bar{\mathbf{H}}} \approx R(\bar{\mathbf{H}}) = Q(\bar{\mathbf{H}})^{-1}P(\bar{\mathbf{H}})$$

–

Implementation: Scalable Library for Eigenvalue Problem Computations (SLEPc)

The sparse matrix computation library we leverage to implement the required matrix exponentiation.

SLEPc:

- Uses PETSc data structures and employs MPI;
- Is available on Cori, straightforward to install locally;
- Has an object-oriented flavor and abstracts away from most MPI subroutines through the use of “collective” library functions;
- Provides scalable building blocks for solving large-scale sparse eigenvalue problems.

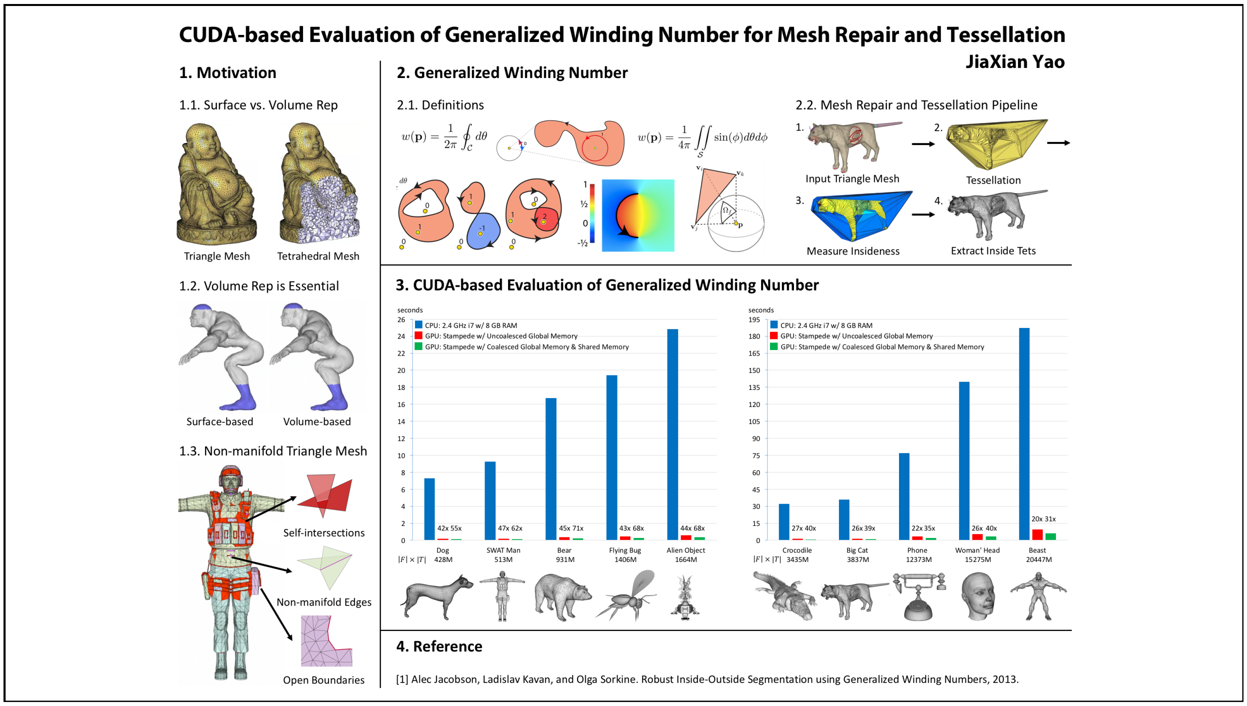
Snowflake

A high-performance eDSL for Stencils in Python

Nathan Zhang

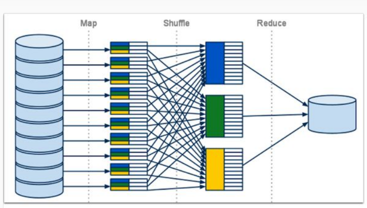
Snowflake

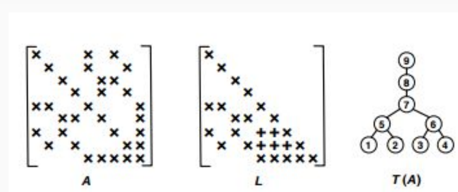
- An eDSL for Stencils in Python
- Performance comparable to Handwritten C/ OpenMP
- Extending to OpenCL
- Can apply high level optimizations and analyses
- **Implemented Communication Avoiding Stencil Decomposition**



Sparse Linear Algebra on Modern Data Processing Systems

Cathy Wu, Philipp Moritz





MapReduce:
- (+) massively parallel
- (+) dynamic scaling of the cluster
- (–) rigid programming model

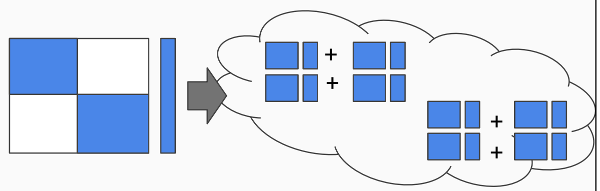
Sparse linear algebra workloads:
- flexible computation patterns
- fine grained tasks

Problem: Sparse Linear Algebra not well supported on modern data processing frameworks like Spark

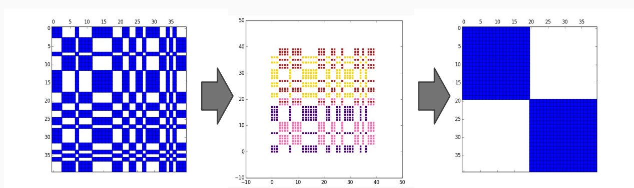
Sparse Linear Algebra on Modern Data Processing Systems

Our project: We investigate how sparse linear algebra can be implemented in next-generation data processing systems

Design principles:
- remotely scheduled fine grained function calls
- remote and distributed objects



Operations:
- Sparse matrix vector multiply
- Cholesky decomposition



Applications:
- Machine learning, e.g. PageRank
- Optimization, e.g. nonlinear programming

Optimizing Fire Simulation with GLSL

Yi Tong
Saurabh Mitra

Problem Statement

Numerical solutions to the Navier-Stokes equations in three dimensions for high-resolution grids are intractably slow

We want to render a fire simulation at or near real-time speeds with overall Navier-Stokes-like behavior

The Solution

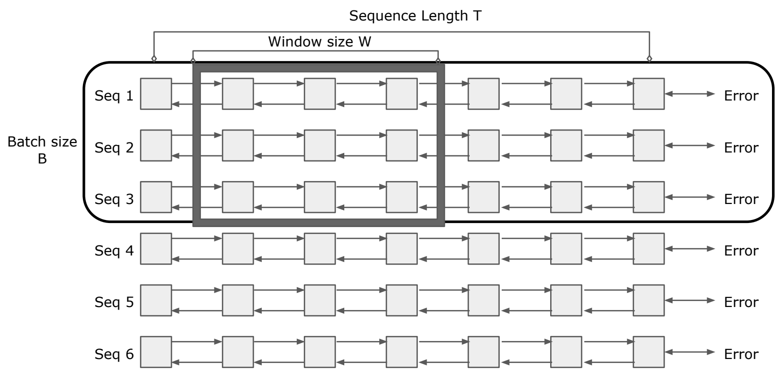
Use a low-resolution Navier-Stokes solver to drive a high volume of particles.

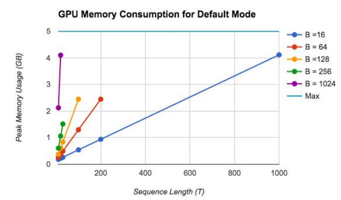
Since particles are already being rendered with OpenGL as small flame sprites, use GLSL Compute Shaders to linearly interpolate particle velocities, temperatures, and densities based on Navier-Stokes grids before updating positions and rendering each frame

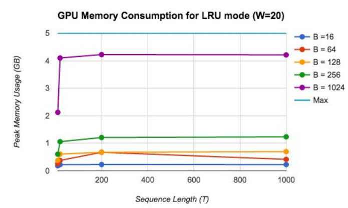
CS 267 Project @ Berkeley [Spr'16] Rohan Padhye

Scaling Long-Sequence RNN training on GPUs

By storing only a fixed window W in GPU memory. Other states swapped to CPU memory on LRU basis.







Increasing T → Better precision
Increasing B → Better parallelism [upto B*]
GPU Memory required: {default: ***O*(TB)**, LRU: ***O*(WB)**}

Fast Parallel Gibbs Sampling on Discrete Bayesian Networks and Factor Graphs

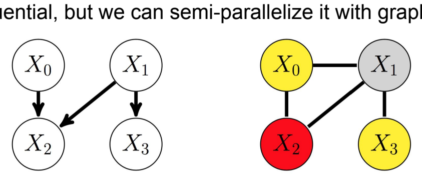
Daniel Seita CS 267 Final Project

Problem Statement

Given partially observed discrete data and a graphical model structure, find the "best" parameters (a.k.a. MAP estimation).

$$\mathcal{D} = \underbrace{\begin{bmatrix} 0 & x & 1 & x & \cdots & 1 \\ 1 & x & 1 & 1 & \cdots & 0 \\ x & 1 & 1 & x & \cdots & 0 \\ 1 & 0 & x & x & \cdots & x \end{bmatrix}}_{\text{Columns = "possible worlds," x = missing data}} \qquad \widetilde{\theta} = \arg\max_{\theta} P(\theta \mid \mathcal{D}) = \arg\max_{\theta} \frac{P(\theta)P(\mathcal{D} \mid \theta)}{P(\mathcal{D})}$$

Gibbs sampling is an MCMC algorithm for sampling the posterior. It is sequential, but we can semi-parallelize it with graph coloring:

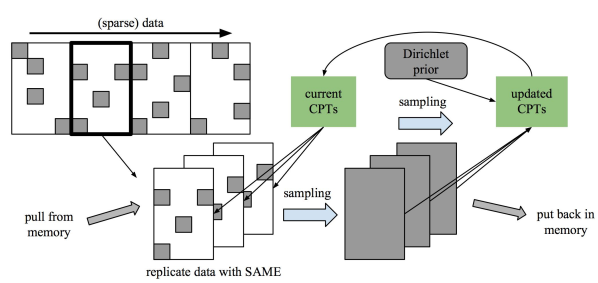


$$X_0 \sim P(X_0 \mid X_1, X_2, X_3) \propto P(X_0)P(X_2 \mid X_1, X_0)$$
$$X_3 \sim P(X_3 \mid X_0, X_1, X_2) \propto P(X_3 \mid X_1)$$

My Implementation

The code is implemented in the BIDMach toolkit. Features:

1. Sampling process is fused into matrix operations
2. GPU kernels for matrix multiplication and other operators
3. Update parameters after each mini-batch of data
4. Data can be as large as disk or network storage space
5. Supports "temperature" for Gibbs sampling (S.A.M.E.)



Future work: more benchmarks (data and code), fix bottlenecks
Code available at: https://github.com/BIDData/BIDMach

GPU Compression - Algorithm

The LZSS algorithm is a compression algorithm that takes advantage of repeated phrases in some text. That is, when a word is repeated within a small frame of one another, the LZSS algorithm replaces the second occurrence of that word with a reference to the first word. For example, we have the phrase:

```
yiwen song is a yiwen song.
```

We can compress to:

```
yiwen song is a (16,10).
```

GPU Compression - Use in HPC

Idea: Compress contents of communication sent over a network.

$c = \alpha + \beta m$

Compression time per Byte $= \kappa$

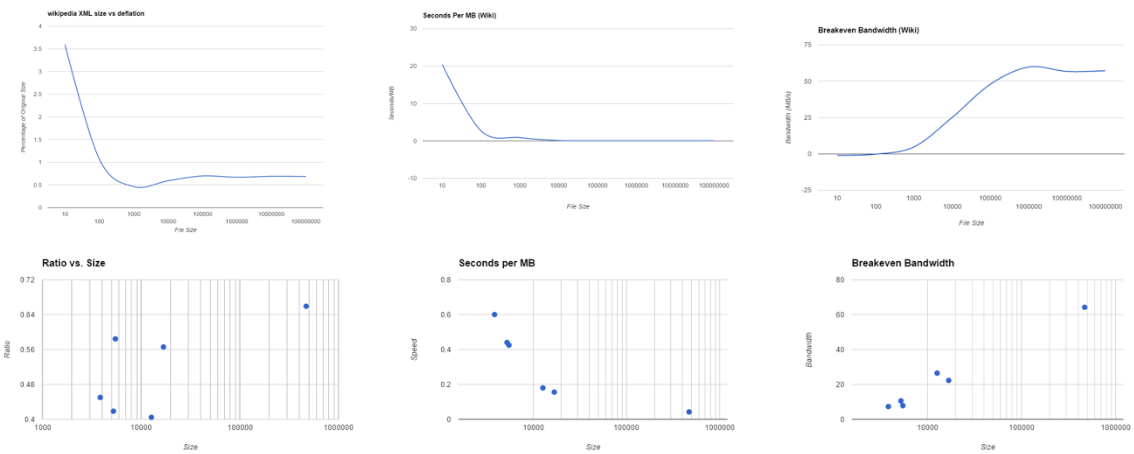
Compressed size to uncompressed size ratio $= D$

$\Rightarrow \alpha + \beta m > \alpha + \beta D m + \kappa m$

$\beta > \dfrac{\kappa}{1 - D}$

$\beta = .08$ for 100Mbps network, $\beta = .32$ for 25Mbps network

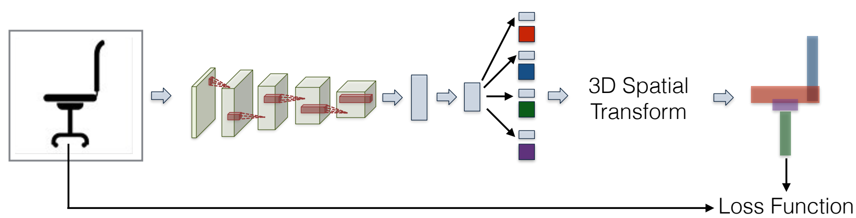
GPU Compression - Performance Data



- FARAZ TAVAKOLI FARAHANI

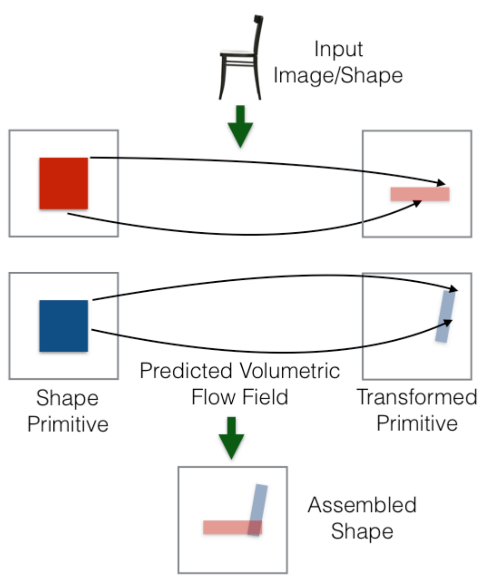
Learning to Assemble Objects from Volumetric Primitives

- Shubham Tulsiani



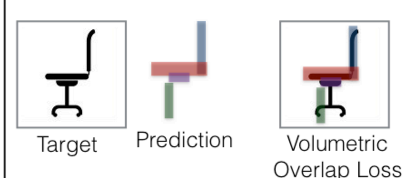
We train a neural network to reconstruct an object by transforming and composing volumetric primitives (cubes). We use a popular deep learning framework 'Torch' and add functionality required for the 3D spatial transforms and loss functions.

3D Spatial Transformer

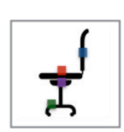


We implement GPU based trilinear interpolation layer to transform shape primitives via flow fields and also yield gradients for training neural networks

Distance Field Computation for Improved Loss Function



The overlap loss function with spatial transformer gives very local gradients and will not correct the mis-aligned green primitive



We add a loss penalizing every unexplained target point's distance to the closest primitive centre

We implement a GPU based distance field computation layer for training neural networks with this additional loss