# A Shared Memory Parallel Algorithm for Data Reduction Using the Singular Value Decomposition

**Rhonda Phillips, Layne Watson, Randolph Wynne**

April 16, 2008

Virginia Tech

# Outline

- Motivation
- Algorithms
- Study Area
- Results and Analysis
- Implementation Details
- Conclusions

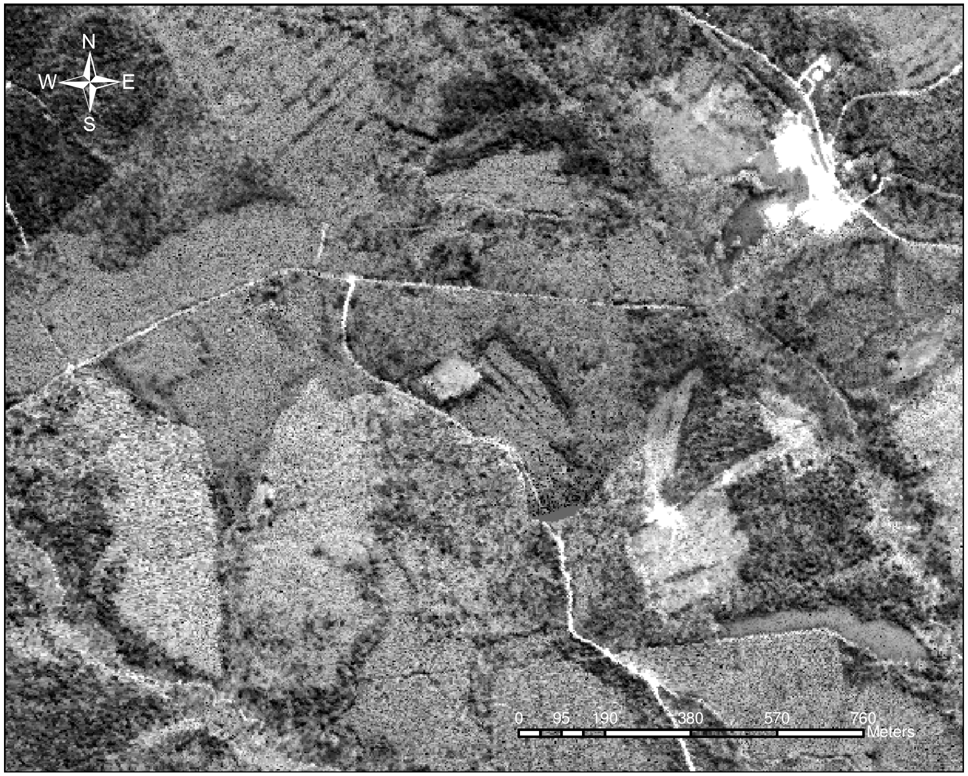
# Motivation

- Satellite images are increasing in size.
    - Spectral and spatial resolutions are increasing.
- Having too many features for the training dataset degrades classification accuracy (overfitting issues).
- The SVD can be expensive to compute, especially for a large dataset.

# SVD-based feature reduction

- $SVDReduce$:
  - Consider $X = U\Sigma V^t$ to be $m \times n$ and $m << n$.
  - In order to represent $X$ using $k$ dimensions ($k < m$), set the singular values in positions $k+1$ to $m$ in $\Sigma$ to zero to form $\tilde{\Sigma}$.
  - $X$ can be represented by coordinates $\tilde{\Sigma}V^t$.
- $SVDTrainingReduce$:
  - Instead of using the image $X$ as in $SVDReduce$, a representative training data set $T$ is used to compute $T = \hat{U}\hat{\Sigma}\hat{V}^t$.
  - $X$ is projected onto the basis set $\hat{U}$.

# Study Area




A hyperspectral image containing 224 bands taken over the Appomattox Buckingham State Forest was used to obtain all execution times listed.

# Execution Times for *SVDReduce*

$U$, the left singular vectors, are computed using the image $X$, and $X$ is projected onto $U$.

| Operation | Time (seconds) |
|---|---:|
| SVD Factorization | 340.04 |
| Matrix-Vector Multiply | 184.33 |
| Other | 8.05 |
| **Total** | **532.42** |

# Execution Times for $SVDTrainingReduce$

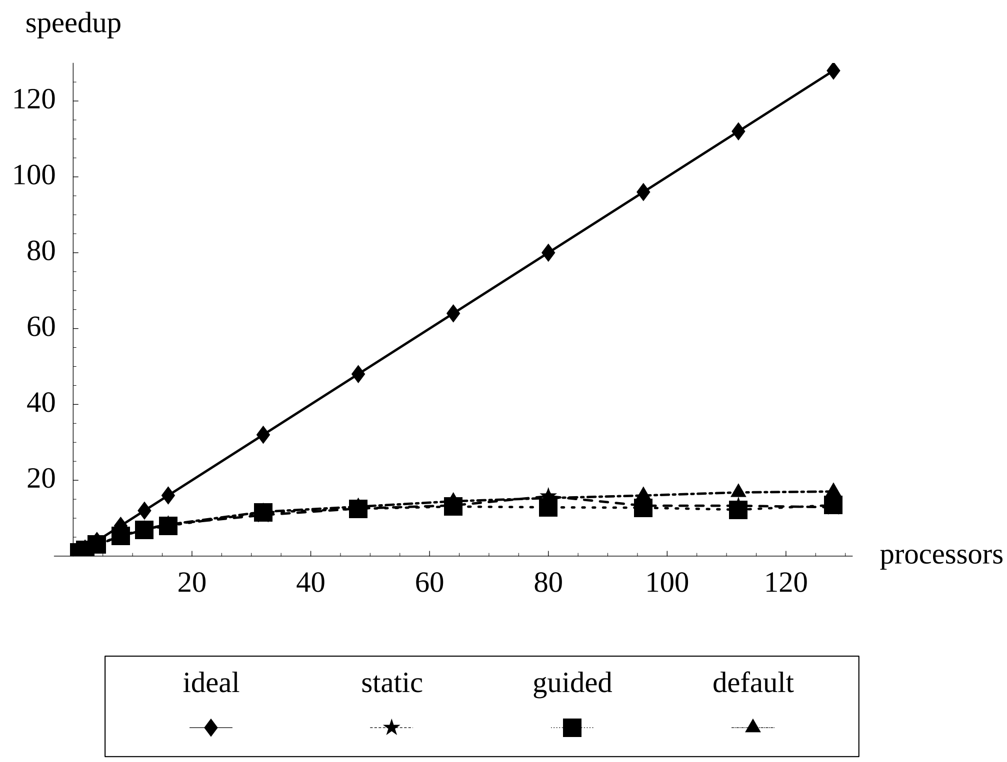
$\hat{U}$, the left singular vectors, are computed using the training data $T$, and $X$ is projected onto $\hat{U}$.

| Operation | Time (seconds) |
| --- | ---: |
| SVD Factorization | .24 |
| Matrix-Vector Multiply | 180.07 |
| Other | 4.71 |
| **Total** | **185.02** |

# Parallel SVD based feature reduction

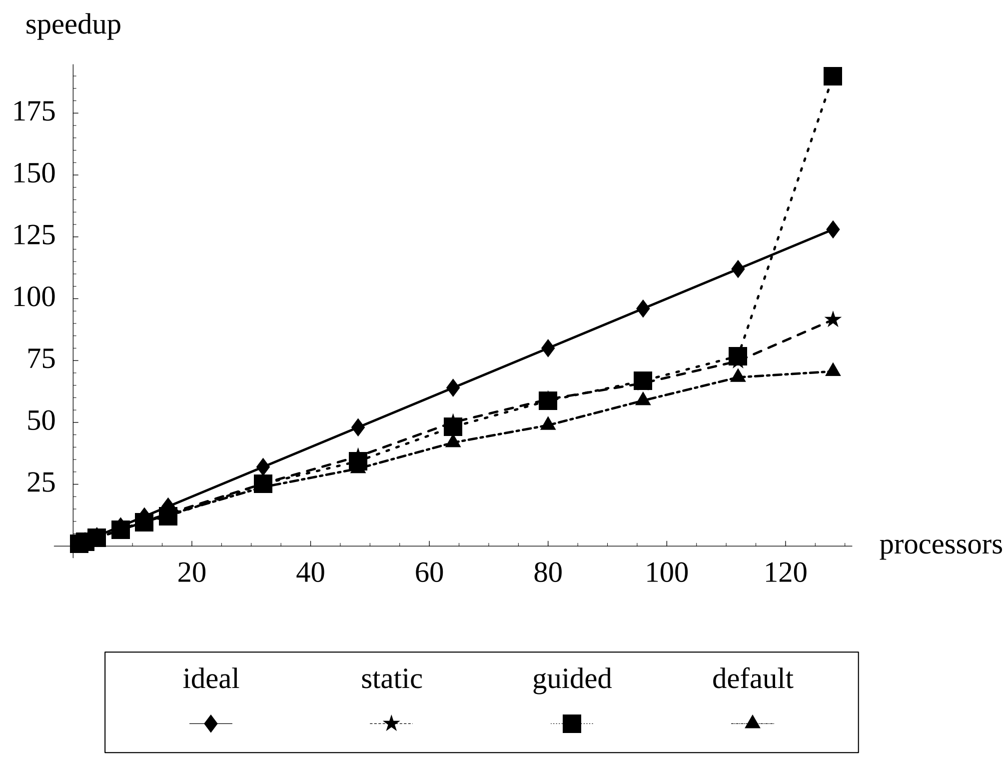
- $SVDTrainingReduce$ is faster than $SVDReduce$, and is more suited to run on a parallel computer.
- The SVD factorization and the projection of $X$ onto $\hat{U}$ can be computed in parallel.
  - The SVD factorization requires much less execution time than projecting $X$ onto $\hat{U}$.
  - In practice, using a (shared memory) parallel SVD factorization was slower than using the serial SVD factorization.

# Parallel Speedup




Parallel speedup of $pSVDTrainingReduce$ including all input/output operations.

# Parallel Speedup



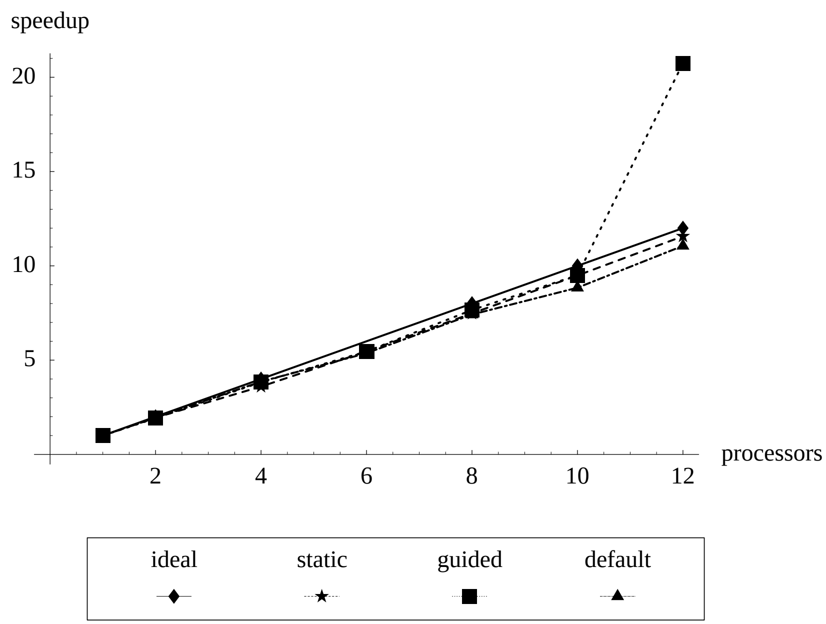

Parallel speedup of $pSVDTrainingReduce$ without input/output execution times included.

# Implementation Issues

- Scheduling
- Data Placement
- Private vs. Shared variables
- Cache

# Scheduling

The speedup increase using all processors can be repeated on smaller SGI Altixes.




Speedup without input/output using 12 processors.

# Scheduling

- Referring to previous speedup graphs, there is an increase in speedup when guided scheduling and all processors are used.
    - Guided scheduling is a type of dynamic scheduling that varies the chunk size.
- This speedup is also observed for dynamic scheduling.
- Although these are dynamic scheduling strategies, data initialization and placement (using `dplace`) is important.
- Allowing the underlying hardware and software to assign work to processors results in a large speedup, but only when using all processors.

# Scheduling

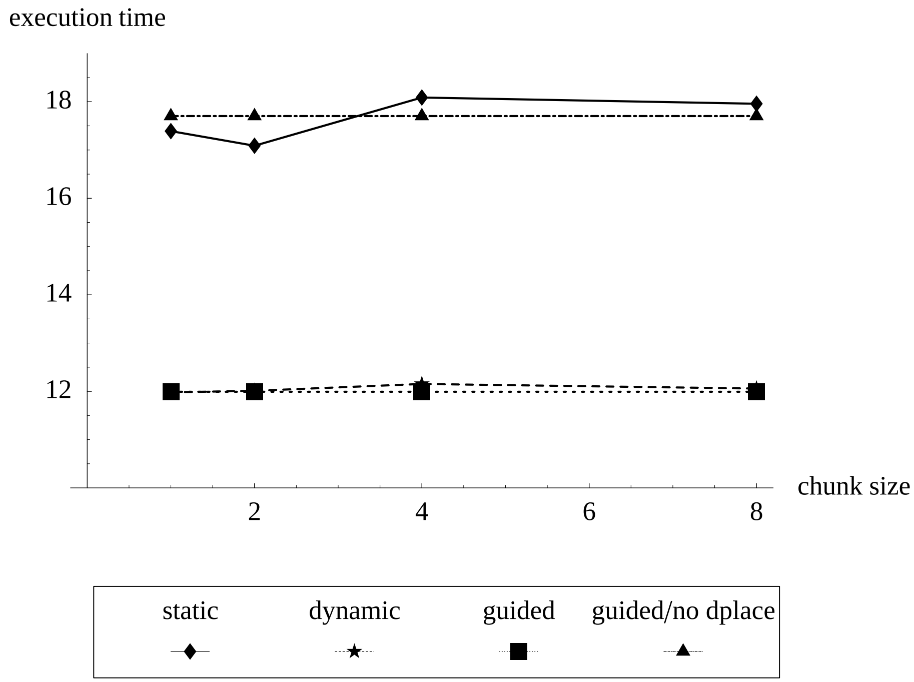
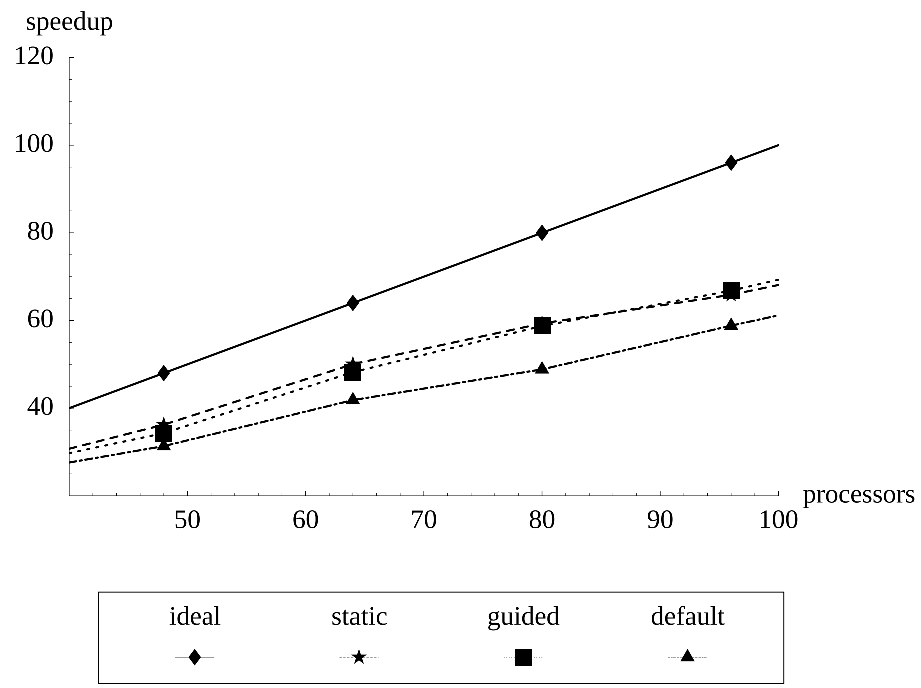



Illustration of execution time differences using various scheduling and data placement strategies when all processors are involved in computation.

# Data Placement




In this zoom of parallel speedup without input/output, the static and guided scheduling strategies using `dplace` and consistent memory access outperform a naive approach.

# Variable Storage

- $U$ is the basis set that every vector in $X$ is projected onto.
- $U$ can be physically stored as static or dynamic memory, and can be private or shared across processors during parallel execution.

Execution times for different methods of storing $U$ using 12 processors.

| Variable Type | Time (seconds) |
|---|---|
| static shared | 20.586 |
| static private | 17.741 |
| dynamic shared | 17.419 |

# Cache optimization

- In the previous slide, cache coherency overhead resulted in increased execution times for static memory that is shared across processors.
- As this algorithm was implemented in Fortran 95, which uses column major order, all data is accessed by column.
    - Each processor then accesses data elements in the order they are laid out in memory, maximizing cache hits.
- The data is also distributed across columns to minimize the likelihood that processors are operating on the same cache line.

# Conclusions

- $pSVDTrainingReduce$ is faster than $SVDTrainingReduce$, which is faster and parallelizes better than $SVDReduce$.
- If input/output are excluded from the execution times, $pSVDTrainingReduce$ scales well to 128 processors since the SVD factorization is a small portion of the algorithm.
- Even for straightforward parallel algorithms such as $pSVDTrainingReduce$, performance tuning is essential.
- There is something interesting happening "under the hood" of the SGI Altix when dynamic scheduling strategies are employed and all processors are utilized.