Efficiently Computing Tensor Eigenvalues on a GPU

Grey Ballard\textsuperscript{1}, Tamara Kolda\textsuperscript{2}, Todd Plantenga\textsuperscript{2}

\textsuperscript{1}UC Berkeley

\textsuperscript{2}Sandia National Laboratories

IPDPS - PDSEC Workshop
May 20, 2011

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.
Motivating application requires solving many small symmetric tensor eigenvalue problems

Recent algorithm was developed for computing symmetric tensor eigenpairs

We show how to exploit symmetry within algorithm for computational efficiency

We provide a GPU implementation that exploits symmetry and parallelism in application
Diffusion weighted magnetic resonance imaging (DW-MRI) can detect nerve fibers in the brain.

Resolving a voxel with one fiber direction is relatively easy.

Resolving crossing fibers is harder, requires more careful measurement and a more difficult computational problem (Schultz & Seidel [SS08]).

Finding Nerve Fiber Directions

- DW-MRI data is used to construct function defined on unit sphere
  - function value represents rate of diffusion in a given direction
  - local maxima correspond to nerve fiber directions

![Image](http://spie.org/x33122.xml?ArticleID=x33122)
Finding Nerve Fiber Directions

- DW-MRI data is used to construct function defined on unit sphere
  - function value represents rate of diffusion in a given direction
  - local maxima correspond to nerve fiber directions

Our goal is to find the critical points of this function (a homogeneous polynomial):

\[
f(x) = \sum_{i_1=1}^{3} \cdots \sum_{i_m=1}^{3} a_{i_1 \cdots i_m} x_{i_1} \cdots x_{i_m}
\]

- This is equivalent to finding symmetric tensor eigenpairs
What is a symmetric tensor eigenpair?

A tensor is a multiway or $m$-way array whose entries can be indexed by $m$ numbers ranging from 1 to $n$.

$m =$ mode/order  $n =$ dimension

Symmetry means $a_{135} = a_{153} = a_{315} = a_{351} = a_{513} = a_{531}$ (when $m = 3$ and $n = 5$ in this example).

λ and $x$ are an eigenpair if $A x^{m-1} = \lambda x$. 

Grey Ballard  
PDSEC Workshop 2011
What is a symmetric tensor eigenpair?

A tensor is a multiway or $m$-way array whose entries can be indexed by $m$ numbers ranging from 1 to $n$.

"Symmetry" means

$$a_{135} = a_{153} = a_{315} = a_{351} = a_{513} = a_{531}$$

($m = 3$ and $n = 5$ in this example)
What is a symmetric tensor eigenpair?

A tensor is a multiway or $m$-way array whose entries can be indexed by $m$ numbers ranging from 1 to $n$.

$m = \text{mode/order} \quad n = \text{dimension}$

"Symmetry" means

$$a_{135} = a_{153} = a_{315} = a_{351} = a_{513} = a_{531}$$

($m = 3$ and $n = 5$ in this example)

$\lambda$ and $x$ are an eigenpair if $x$ has unit length and $A x^{m-1} = \lambda x$
Computing Tensor Eigenpairs

We can compute tensor eigenpairs using a shifted symmetric higher order power method: SS-HOPM (Kolda & Mayo [KM10])

**SS-HOPM**

\[
\begin{align*}
\text{repeat} & \\
\mathbf{x}_{k+1} & \leftarrow \frac{\mathcal{A}\mathbf{x}_k^{m-1} + \alpha \mathbf{x}_k}{\| \mathcal{A}\mathbf{x}_k^{m-1} + \alpha \mathbf{x}_k \|} \\
\lambda_{k+1} & \leftarrow \mathcal{A}\mathbf{x}_k^m \\
\text{until } & \lambda_k \text{ converges}
\end{align*}
\]

**Power Method**

\[
\begin{align*}
\text{repeat} & \\
\mathbf{x}_{k+1} & \leftarrow \frac{\mathbf{A}\mathbf{x}_k}{\| \mathbf{A}\mathbf{x}_k \|} \\
\lambda_{k+1} & \leftarrow \mathbf{x}_{k+1}^T \mathbf{A}\mathbf{x}_{k+1} \\
\text{until } & \lambda_k \text{ converges}
\end{align*}
\]
Computing Tensor Eigenpairs

We can compute tensor eigenpairs using a shifted symmetric higher order power method: SS-HOPM (Kolda & Mayo [KM10])

\[ \text{SS-HOPM} \]
\[
\begin{align*}
\text{repeat} & \\
x_{k+1} & \leftarrow \frac{Ax_{k}^{m-1} + \alpha x_{k}}{\|Ax_{k}^{m-1} + \alpha x_{k}\|} \\
\lambda_{k+1} & \leftarrow Ax_{k+1}^{m} \\
\text{until} & \lambda_{k} \text{ converges}
\end{align*}
\]

\[ \text{Power Method} \]
\[
\begin{align*}
\text{repeat} & \\
x_{k+1} & \leftarrow \frac{Ax_{k}}{\|Ax_{k}\|} \\
\lambda_{k+1} & \leftarrow x_{k+1}^{T}Ax_{k+1} \\
\text{until} & \lambda_{k} \text{ converges}
\end{align*}
\]

- SS-HOPM always converges with appropriate shift \( \alpha \)
- Different starting vectors \( x_{0} \) may converge to different eigenvectors
- Other differences from matrix case:
  - eigenpairs not all real (though there are always at least two)
  - eigenvectors not all orthogonal
A symmetric tensor in $\mathbb{R}^{[m,n]}$ has $\binom{m+n-1}{m}$ unique entries.
Exploiting Symmetry

Data Movement/Storage

A symmetric tensor in $\mathbb{R}^{[m,n]}$ has $\binom{m+n-1}{m}$ unique entries.

$n^m \rightarrow \frac{n^m}{m!} + O(n^{m-1})$

$m = \text{mode/order} \quad n = \text{dimension}$

Computation

Main kernels: $\mathcal{A}x^{m-1}$ and $\mathcal{A}x^m$

SS-HOPM

Repeat

$x_{k+1} \leftarrow \frac{\mathcal{A}x_k^{m-1} + \alpha x_k}{\|\mathcal{A}x_k^{m-1} + \alpha x_k\|}$

$\lambda_{k+1} \leftarrow \mathcal{A}x_{k+1}^m$

Until $\lambda_k$ converges
\( A x^m \) Kernel

“tensor times same vector in all modes”

Non-symmetric

\[
A x^m = \sum_{i_1=1}^{n} \cdots \sum_{i_m=1}^{n} a_{i_1 \ldots i_m} x_{i_1} \cdots x_{i_m}
\]

- sequence of matrix-vector products
- cost: \( 2n^m + O(n^{m-1}) \)
**Ax^m Kernel**

"tensor times same vector in all modes"

**Non-symmetric**

\[ Ax^m = \sum_{i_1=1}^{n} \cdots \sum_{i_m=1}^{n} a_{i_1 \ldots i_m} x_{i_1} \cdot \cdots \cdot x_{i_m} \]

- sequence of matrix-vector products
- cost: \(2n^m + O(n^{m-1})\)

**Symmetric**

\[ Ax^m = \sum_{I \subseteq [m,n]} \binom{m}{k_1, k_2, \ldots, k_n} a_{i_1 \ldots i_m} x_1^{k_1} \cdots x_n^{k_n} \]

- \(k_i\) is number of occurrences of \(i\) in \(I = [i_1, \ldots, i_m]\)
- cost: \(O\left(\frac{n^m}{(m-1)!}\right)\)
$Ax^m$ Kernel

“tensor times same vector in all modes”

Non-symmetric

$Ax^m = \sum_{i_1=1}^{n} \ldots \sum_{i_m=1}^{n} a_{i_1 \ldots i_m} x_{i_1} \ldots x_{i_m}$

sequence of matrix-vector products

cost: $2n^m + O(n^{m-1})$

Symmetric

$Ax^m = \sum_{l \in j^{[m,n]}} \binom{m}{k_1, k_2, \ldots, k_n} a_{i_1 \ldots i_m} x_{i_1}^{k_1} \ldots x_{i_m}^{k_n}$

$k_i$ is number of occurrences of $i$ in $l = [i_1, \ldots, i_m]$

cost: $O\left(\frac{n^m}{(m-1)!}\right)$

$Ax^{m-1}$ kernel is slightly more complicated, but same idea
Mapping SSHOPM to GPU

- In our application, we have many small tensor eigenproblems
  - $256 \times 170 \times 170$ voxels per scan
  - each voxel has $3 \times 3 \times 3 \times 3$ symmetric tensor ($m = 4, n = 3$)

- Must try multiple starting vectors for each tensor

- Lots of independent problems

We assign...
Computing Tensor Eigenpairs

We can compute tensor eigenvalues and vectors using a shifted symmetric higher order power method (SS-HOPM) \[KM10\]

SS-HOPM always converges with appropriate shift

Other differences from matrix case:
- Different starting vectors
- Eigenvectors not all orthogonal
- Eigenpairs not all real (though there are always at least two)

Must try multiple starting vectors for each tensor

Lots of independent problems

We assign
- One thread block per tensor
- One thread per starting vector

In our application, we have many small tensor eigenproblems
- \(256 \times 170 \times 170\) voxels per scan
- Each voxel has \(3 \times 3 \times 3 \times 3\) symmetric tensor \((m = 4, n = 3)\)
Implementation

Most important optimization: do symmetric bookkeeping at compile-time and completely unroll loops

- tensor indices and multinominal coefficients stored in instructions
- tensors stored in shared memory
- vectors stored in registers

\[ y_2 = A_{\text{vals}[1]} \times x_1 \times x_1 \times x_1 + A_{\text{vals}[3]} \times 3 \times x_1 \times x_1 \times x_2 + A_{\text{vals}[4]} \times 3 \times x_1 \times x_1 \times x_3 + A_{\text{vals}[6]} \times 3 \times x_1 \times x_2 \times x_2 + A_{\text{vals}[7]} \times 6 \times x_1 \times x_2 \times x_3 + A_{\text{vals}[8]} \times 3 \times x_1 \times x_3 \times x_3 + A_{\text{vals}[10]} \times x_2 \times x_2 \times x_2 + A_{\text{vals}[11]} \times 3 \times x_2 \times x_2 \times x_3 + A_{\text{vals}[12]} \times 3 \times x_2 \times x_3 \times x_3 + A_{\text{vals}[13]} \times x_3 \times x_3 \times x_3; \]
Implementation

Most important optimization: do symmetric bookkeeping at compile-time and completely unroll loops

- tensor indices and multinomial coefficients stored in instructions
- tensors stored in shared memory
- vectors stored in registers

\[
y^2 = \text{Avals}[1] \cdot x_1 \cdot x_1 \cdot x_1 + \\
\text{Avals}[3] \cdot 3 \cdot x_1 \cdot x_1 \cdot x_2 + \\
\text{Avals}[4] \cdot 3 \cdot x_1 \cdot x_1 \cdot x_3 + \\
\text{Avals}[6] \cdot 3 \cdot x_1 \cdot x_2 \cdot x_2 + \\
\text{Avals}[7] \cdot 6 \cdot x_1 \cdot x_2 \cdot x_3 + \\
\text{Avals}[8] \cdot 3 \cdot x_1 \cdot x_3 \cdot x_3 + \\
\text{Avals}[10] \cdot x_2 \cdot x_2 \cdot x_2 + \\
\text{Avals}[11] \cdot 3 \cdot x_2 \cdot x_2 \cdot x_3 + \\
\text{Avals}[12] \cdot 3 \cdot x_2 \cdot x_3 \cdot x_3 + \\
\text{Avals}[13] \cdot x_3 \cdot x_3 \cdot x_3;
\]

- use code generator
- need to know mode and dimension at compile time
- assumes vectors fit in register file
Results

Synthetic test problem of $32 \times 32$ tensors, includes a fiber crossing

- from SCI Institute, U. of Utah

- Tesla C 2050 (Fermi)
- single precision
- unrolled code: 31% of peak
Future Work

- Mathematical approach to DW-MRI application may be flawed
  - may need higher order tensors to resolve small crossing angles
  - may need symmetric low-rank approximation instead of eigenpairs

- Consider more difficult cases
  - larger problems - data doesn’t fit on-chip
  - only one tensor (less trivial parallelism)
  - only one vector (all threads work together on one iteration)

- Consider other algorithms
  - What other tensor algorithms could exploit symmetry?
Thank you!

Grey Ballard
ballard@cs.berkeley.edu
www.eecs.berkeley.edu/~ballard

Tamara Kolda        Todd Plantenga

Sandia National Laboratories
References

Tamara G. Kolda and Jackson R. Mayo.
Shifted power method for computing tensor eigenpairs.

Thomas Schultz and Hans-Peter Seidel.
Estimating crossing fibers: A tensor decomposition approach.