Toward an Automatically Tuned Dense Symmetric Eigensolver for Shared Memory Machines

Yusaku Yamamoto
Dept. of Computational Science & Engineering
Nagoya University
Outline of the talk

1. Introduction
2. High performance tridiagonalization algorithms
3. Performance comparison of two tridiagonalization algorithms
4. Optimal choice of algorithm and its parameters
5. Conclusion
Introduction

• Target problem
  – Standard eigenvalue problem \( Ax = \lambda x \)
    • \( A \) : \( n \) by \( n \) real symmetric matrix
    • We assume that \( A \) is dense.

• Applications
  – Molecular orbital methods
    • Solution of dense eigenproblems of order more than 100,000 is required to compute the electronic structure of large protein molecules.

  – Principal component analysis
Target machines

- Symmetric Multi-Processors (SMP’s)

- Background
  - PC servers with 4 to 8 processors are quite common now.
  - The number of CPU cores integrated into one chip is rapidly increasing.

- dual-core Xeon

- Cell processor (1+8 cores)
Flow chart of a typical eigensolver for dense symmetric matrices

Real symmetric $A$ 

- Tridiagonalization

Tridiagonal matrix $T$

- Eigensolution of the tridiagonal matrix

  - Eigenvalues of $T$: $\{\lambda_i\}$
  - Eigenvectors of $T$: $\{y_i\}$

Back transformation

Eigenvectors of $A$: $\{x_i\}$

Computation

- $Q^*AQ = T$
  - ($Q$: orthogonal)

- $Ty_i = \lambda_i y_i$

Algorithm

- Householder method
- QR method
- D & C method
- Bisection & IIM
- MR$^3$ algorithm
- Back transformation
Objective of this study

• Study the performance of tridiagonalization and back transformation algorithms on various SMP machines.
  – Dongarra’s algorithm
  – Bischof’s algorithm

• The optimal choice of algorithm and its parameters will differ greatly depending on the computational environment and the problem size.

• The obtained data will be useful in designing an automatically tuned eigensolver for SMP machines.
Algorithms for tridiagonalization and back transformation

- Tridiagonalization by the Householder method
  - Reduction by Householder transformation $H = I - \alpha uu^t$
    - $H$ is a symmetric orthogonal matrix.
    - $H$ eliminates all but the first element of a vector.
    - $HA^{(k)} = A^{(k)} - \alpha u (u^t A^{(k)})$

- Computation at the $k$-th step

- Back transformation
  - Apply the Householder transformations in reverse order.
Problems with the conventional Householder method

• Characteristics of the algorithm
  – Almost all the computational work are done in matrix vector multiplication and rank-2 update, both of which are level-2 BLAS.
    • Total computational work: \((4/3)n^3\)
    • Matrix-vector multiplication: \((2/3)n^3\)
    • rank-2 update: \((2/3)n^3\)

• Problems
  – Poor data reuse due to the level-2 BLAS
    • The effect of cache misses and memory access contentions among the processors is large.
  – Cannot attain high performance on modern SMP machines.
High performance tridiagonalization algorithms (I)

• Dongarra’s algorithm (Dongarra et al., 1992)
  – Defer the application of the Householder transformations until several \( M \) transformations are ready.
  – Rank-2 update can be replaced by rank-2\( M \) update, enabling a more efficient use of cache memory.
  – Adopted by LAPACK, ScaLAPACK and other libraries.

\[
\begin{bmatrix}
0 \\
0 \\
A^{(K\times M)}
\end{bmatrix} =
\begin{bmatrix}
U^{(K\times M)} \\
Q^{(K\times M)^t} \\
Q^{(K\times M)} \\
U^{(K\times M)^t}
\end{bmatrix} 
\times
\begin{bmatrix}
\times \\
\times \\
\times \\
\times
\end{bmatrix}
\]

– rank-2\( M \) update (using level-3 BLAS)

• Back transformation
  – Can be blocked in the same manner to use the level-3 BLAS.
Properties of Dongarra’s algorithm

• Computational work
  – Tridiagonalization: \((4/3)n^3\)
  – Back transformation: \(2n^2m\) \((m: \text{number of eigenvectors})\)

• Parameters to be determined
  – \(M_T\) (block size for tridiagonalization)
  – \(M_B\) (block size for back transformation)

• Level-3 aspects
  – In the tridiagonalization phase, only half of the total work can be done with the level-3 BLAS.
  – The other half is level-2 BLAS, thereby limiting the performance on SMP machines.
High performance tridiagonalization algorithms (II)

- Two-step tridiagonalization (Bishof et al., 1993, 1994)
  - First, transform $A$ into a band matrix $B$ (of half bandwidth $L$).
    - This can be done almost entirely with the level-3 BLAS.
  - Next, transform $B$ into a tridiagonal matrix $T$.
    - The work is $O(n^2L)$ and is much smaller than that in the first step.

![Diagram showing the transformation process]

- Transformation into a band matrix
High performance tridiagonalization algorithms (II)

• Wu’s modification (Wu et al., 1996)
  – Defer the application of the block Householder transformations until several ($M_T$) transformations are ready.
  – Rank-2$L$ update can be replaced by rank-2$L M_T$ update, enabling a more efficient use of cache memory.

• Back transformation
  – Two-step back transformation is necessary.
    • 1$^\text{st}$ step: eigenvectors of $T$ -> eigenvectors of $B$
    • 2$^\text{nd}$ step: eigenvectors of $B$ -> eigenvectors of $A$
  – Both steps can be reorganized to use the level-3 BLAS.
  – In the 2$^\text{nd}$ step, several ($M_B$) block Householder transformations can be aggregated to enhance cache utilization.
Properties of Bischof’s algorithm

• Computational work
  – Reduction to a band matrix: \((4/3)n^3\)
  – Murata’s algorithm: \(6n^2L\)
  – Back transformation
    • 1st step: \(2n^2m\)
    • 2nd step: \(2n^2m\)

• Parameters to be determined
  – \(L\) (half bandwidth of the intermediate band matrix)
  – \(M_T\) (block size for tridiagonalization)
  – \(M_B\) (block size for back transformation)

• Level-3 aspects
  – All the computations that require \(O(n^3)\) work can be done with the level-3 BLAS.
We compare the performance of Dongarra’s and Bischof’s algorithm under the following conditions:

- **Computational environments**
  - Xeon (2.8GHz)
  - Fujitsu PrimePower HPC2500
  - Opteron (1.8GHz)
  - IBM pSeries (Power5, 1.9GHz)

- **Number of processors**
  - 1 to 32 (depending on the target machines)

- **Problem size**
  - \( n = 1500, 3000, 6000, 12000 \) and 24000
  - All eigenvalues & eigenvectors, or eigenvalues only

- **Parameters in the algorithm**
  - Choose nearly optimal values for each environment and matrix size.

- **SMP parallelization**
  - Use a parallel version of BLAS.
Bischof's algorithm is preferable when only the eigenvalues are needed.
Performance on the Opteron (1.8GHz) processor

Bischof’s algorithm is preferable for the 4 processor case even when all the eigenvalues and eigenvectors are needed.
Performance on the Fujitsu PrimePower HPC2500

**Execution time (sec.)**

- **$n = 6000$**
- **$n = 12000$**
- **$n = 24000$**

Time for computing all the eigenvalues and eigenvectors

- Bischof’s algorithm is preferable when the number of processors is greater than 2 even for computing all the eigenvectors.
Performance on the IBM pSeries (Power5, 1.9GHz)

Time for computing all the eigenvalues and eigenvectors

Time for computing all the eigenvalues only

Bischof’s algorithm is preferable when only the eigenvalues are needed.
Optimal algorithm as a function of the number of processors and necessary eigenvectors

- **Xeon (2.8GHz)**
  - $n = 1500$
  - $n = 3000$
  - $n = 6000$

- **Opteron (1.8GHz)**
  - $n = 1500$
  - $n = 3000$
  - $n = 6000$
Optimal algorithm as a function of the number of processors and necessary eigenvectors (Cont’d)

- **PrimePower HPC2500**

  ![Graph showing the fraction of eigenvectors needed for different numbers of processors and eigenvectors.](image)

- **pSeries**

  ![Graph showing the fraction of eigenvectors needed for different numbers of processors and eigenvectors.](image)
Optimal choice of algorithm and its parameters

- To construct an automatically tuned eigensolver, we need to select the optimal algorithm and its parameters according to the computational environment and the problem size.

- To do this efficiently, we need an accurate and inexpensive performance model that predicts the performance of an algorithm given the computational environment, problem size and the parameter values.

- A promising way to achieve this goal seems to be the hierarchical modeling approach (Cuenca et al., 2004):
  - Model the execution time of BLAS primitives accurately based on the measured data.
  - Predict the execution time of the entire algorithm by summing up the execution times of the BLAS primitives.
Performance prediction of Bischof’s algorithm

- Predicted and measured execution time of reduction to a band matrix (Opteron, \( n=1920 \))

- The model reproduces the variation of the execution time as a function of \( L \) and \( M_T \) fairly well.
- Prediction errors: less than 10%
- Prediction time: 0.5s (for all the values of \( L \) and \( M_T \))
Automatic optimization of parameters $L$ and $M_T$

- Performance for the optimized values of $(L, M_T)$

The values of $(L, M_T)$ determined from the model were actually optimal for almost all cases.

Parameter optimization in other phases and selection of the optimal algorithm is in principle possible using the same methodology.
Conclusion

• We studied the performance of Dongarra’s and Bischof’s tridiagonalization algorithms on various SMP machines for various problem sizes.

• The results show that the optimal algorithm strongly depends on the target machine, the number of processors and the number of necessary eigenvectors. On the Opteron and HPC2500, Bischof’s algorithm is always preferable when the number of processors exceeds 2.

• By using the hierarchical modeling approach, it seems possible to predict the optimal algorithm and its parameters prior to execution. This will open a way to an automatically tuned eigensolver.