Parallelization of the Molecular Orbital Program MOS-F

Akira Asato, Satoshi Onodera, Yoshie Inada, Elena Akhmatskaya, Ross Nobes, Azuma Matsuura, Atsuya Takahashi

November 2003
Plan of talk

- Introduction to MOS-F
- Parallelizing MOS-F
- Choice of eigensolver
- Performance evaluation on an IA32 cluster
- Conclusions
Introduction to MOS-F

MOS-F is a semiempirical molecular orbital package for spectroscopy developed by A. Matsuura of Fujitsu Laboratories

Main Features

- Semiempirical molecular orbital methods INDO/S, CNDO/S, CNDO/S2, CNDO/S3 and CNDO/2
- Calculations of
  - UV-visible spectrum
  - Electron density and dipole moment in excited states
  - Electron density and dipole moment difference between the ground and the excited state
  - Frequency-dependent polarizability ($\alpha$), first hyperpolarizability ($\beta$) and second hyperpolarizability ($\gamma$)
- Coordinate input from Gaussian Z-matrix or MOPAC internal coordinates
- WinMOPAC IO interface
Introduction to MOS-F: Computational limitations

- Large memory requirements: $\sim n^2$ ($n$ is number of basis functions)
- Calculations restricted to relatively small molecules (currently $n \leq 5,000$)
- $n$ must be tens of thousands to develop new materials or pharmaceutical products

For large systems: Data distribution is needed

Two major components of the computational time for typical MOS-F calculations:
- Determination of all eigenvalues and eigenvectors of a real symmetric matrix
- Matrix multiplication

To perform MOS-F calculations efficiently: Optimized eigensolver and matrix multiplication routines are required

**Computational profile of crambin, $n=1,622$**
- Matrix multiplication: 62.4%
- Eigensolver: 37.1%

**Computational profile of poly-azomethyne, $n=807$**
- Eigensolver: 85.8%
Introduction to MOS-F: Computational limitations

- Large memory requirements: $\sim n^2$ (\(n\) is number of basis functions)
- Calculations restricted to relatively small molecules (currently \(n \leq 5,000\))
- \(n\) must be tens of thousands to develop new materials or pharmaceutical products

For large systems: Data distribution is needed

Two major components of the computational time for typical MOS-F calculations:
- Determination of all eigenvalues and eigenvectors of a real symmetric matrix
- Matrix multiplication

To perform MOS-F calculations efficiently: Optimized eigensolver and matrix multiplication routines are required

Computational profile of crambin, \(n=1,622\)
- Matrix multiplication: 62.4%
- Eigensolver: 37.1%

Computational profile of poly-azomethyne, \(n=807\)
- Eigensolver: 85.8%
Parallelization approaches

Two different approaches to parallelizing MOS-F:

*Introduce parallelism through a set of localized code changes, with ScaLAPACK eigensolvers and matrix multiplication being used (LP-MOS-F)*

- simple implementation
- high single-processor performance
- but, substantial memory requirements

*Overall data distribution approach (DD-MOS-F)*

- memory usage is reduced significantly
- both computational work and memory scale with number of processors

Required Memory for Crambin (n=1622) with DD-MOS-F
**Choice of eigensolver: LAPACK**

**Eigensolvers used in the testing procedure**

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix storage</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS-F eigensolver (EI SPACK)</td>
<td>Packed</td>
<td>Rational QL + inverse iteration</td>
</tr>
<tr>
<td>DSY(P)EV</td>
<td>Full (Packed)</td>
<td>Implicitly shifted QR</td>
</tr>
<tr>
<td>DSY(P)EVX</td>
<td>Full (Packed)</td>
<td>Implicitly shifted QR or bisection + inverse iteration</td>
</tr>
<tr>
<td>DSY(P)EVD</td>
<td>Full (Packed)</td>
<td>Divide and conquer (DC)</td>
</tr>
<tr>
<td>DSYEVR</td>
<td>Full</td>
<td>Relatively robust representations (RRR)</td>
</tr>
</tbody>
</table>

**Use of packed storage results in large increase in CPU time over full storage**

**Two LAPACK eigensolvers RRR and DC show consistently better performance than the original MOS-F code**

---

**Comparative performance of LAPACK solvers**

Input matrix: 4225 x 4225  
Machine environment: 733 MHz Pentium III
**Choice of eigensolver: ScaLAPACK**

**Parallel eigensolvers** used in the testing procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Matrix storage</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDSYEV</td>
<td>Full</td>
<td>Implicitly shifted QR</td>
</tr>
<tr>
<td>PDSYEVX</td>
<td>Full</td>
<td>Bisection + inverse iteration</td>
</tr>
<tr>
<td>PDSYEVD</td>
<td>Full</td>
<td>Divide and conquer (DC)</td>
</tr>
</tbody>
</table>

The parallel QR eigensolver PDSYEV is not competitive in terms of parallel performance with the other parallel eigensolvers.

There is little to choose between PDSYEVX and PDSYEVD on the basis of wall time performance.

**Parallel performance of ScaLAPACK eigensolvers**

**Input Matrix:** 4225 x 4225  
**Machine environment:** A cluster of four dual-processor 733 MHz Pentium III nodes connected via Myrinet.
Choice of eigensolver: ScaLAPACK

- For inverse iteration methods, eigenvalues that are close in value are “clustered” onto the same processor
  - Orthogonalization of the set of eigenvectors corresponding to these eigenvalues can then be performed without communication
  - This can lead to load imbalance and hence poor parallel scaling
- Parallel performance of PDSYEVX is therefore sensitive to the choice of orthogonalization tolerance parameter ORFAC
- The choice of ORFAC to minimize clustering but at the same time retain acceptable accuracy is not trivial
- For this reason, divide and conquer methods are “safer”

Effect of Reorthogonalisation on Scaling of PDSYEVX
Choice of eigensolver: Memory requirements

The memory required for the fast parallel eigensolvers is much greater than for the current MOS-F code.

**PDSYEVX**: Requires a fixed $N^2$ array on each processor to ensure that orthogonalisation of eigenvectors is not limited by available workspace.

**PDSYEVD**: Memory requirement is high but scales with increasing processor count due to completely distributed data.

Our choice: Use DSYEVR (relatively robust representation) on a single node and PDSYEVD (divide and conquer) in parallel.
### Parallelization approaches: Key features

<table>
<thead>
<tr>
<th>Features</th>
<th>DD-MOS-F</th>
<th>LP-MOS-F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requires separate source code</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Standard mathematical libraries</td>
<td>BLAS, ScaLAPACK</td>
<td>BLAS, LAPACK, ScaLAPACK</td>
</tr>
<tr>
<td>Linear algebra routines used</td>
<td>PDSYEVD (block size=1)</td>
<td>DSYEVR, PDSYEVD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DGEMM, PDGEMM</td>
</tr>
<tr>
<td>Data distribution</td>
<td>Overall</td>
<td>Local</td>
</tr>
<tr>
<td>Memory use compared with original MOS-F</td>
<td>Lower</td>
<td>Higher</td>
</tr>
</tbody>
</table>

- A block size of 1 is used in DD-MOS-F. This is compatible with the data distribution scheme used in the code, but is not optimal in terms of computational performance.

- The optimal block size can be used in LP-MOS-F at the expense of higher memory usage (due to local data redistribution).
Performance evaluation: Machine environment

FLE Linux IA Cluster

- Front-end server
  Dual 2.4 GHz Pentium Xeon, 1 GB memory, 120 GB disk
  Intel and Portland Group compilers and DDT parallel debugger

- Gigabit Ethernet switch

- Compute nodes
  Dual 2.4 GHz Pentium Xeon with 2 GB memory

- Myrinet 2000 interconnect

- Compute nodes
  Dual 2.4 GHz Pentium Xeon with 2 GB memory

- Compute nodes
  Dual 2.4 GHz Pentium Xeon with 2 GB memory

- Compute nodes
  Dual 2.4 GHz Pentium Xeon with 2 GB memory
## Performance evaluation: Benchmarks

<table>
<thead>
<tr>
<th>Job</th>
<th>Formula</th>
<th>Number of Atoms</th>
<th>Number of Basis Functions</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>crambin</td>
<td>C_{202}H_{314}N_{55}O_{64}S_{6}</td>
<td>641</td>
<td>1622</td>
<td>CNDO/ S SCF</td>
</tr>
<tr>
<td>Poly-azomethyne (azm)</td>
<td>C_{153}H_{111}N_{21}</td>
<td>285</td>
<td>807</td>
<td>CNDO/ SCi (80,80)</td>
</tr>
</tbody>
</table>
**Performance evaluation: Serial performance**

*LP-MOS-F outperforms DD-MOS-F on a single node due to the use of square rather than packed matrix storage combined with fast LAPACK and BLAS libraries.*

**DD-MOS-F vs LP-MOS-F. Comparative Single-Processor Performance**

**Compiler:** pgf90 using `–fast –tp p7 –Mdalign` optimization

LP-MOS-F uses **LAPACK 3.0** and **Intel MKL 5.2** for BLAS
Performance evaluation: Parallel performance

Parallel performance was measured on the IA32 cluster using different MPI implementations and interconnects:

- **LAM 6.5.9**
- **MPI CH 1.2.4**
- **MPI CH 1.2.4** implemented under **SCore 5.4.0** using **Gigabit Ethernet** network
- **MPI CH 1.2.4** implemented under **SCore 5.4.0** using **Myrinet 2000** network
- **MPI CH-GM 1.2.4..8a** under **GM 1.6**

**PGI** compilers (pgf77, pgf90) with **-fast -tp p7 -Mdalign** optimization were used in all cases
Performance evaluation using various MPI implementations

Parallel Scaling of DD-MOS-F. Poly-azomethyne
Performance evaluation: Parallel scaling of crambin benchmark

DD-MOS-F vs LP-MOS-F. Parallel Scaling.
Crambin

<table>
<thead>
<tr>
<th>Speed-up</th>
<th>0.00</th>
<th>2.00</th>
<th>4.00</th>
<th>6.00</th>
<th>8.00</th>
<th>10.00</th>
<th>12.00</th>
<th>14.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

- DD-MOS-F Myrinet
- DD-MOS-F Ethernet
- DD-MOS-F LAM
- LP-MOS-F Myrinet
- LP-MOS-F Ethernet
- LP-MOS-F LAM
- Ideal
Performance evaluation: Parallel performance of crambin benchmark

DD-MOS-F vs LP-MOS-F. Parallel Performance.
Crambin. SCore Myrinet
Performance evaluation: Parallel scaling of poly-azomethyne benchmark

DD-MOS-F vs LP-MOS-F. Parallel Scaling.
Poly-azomethyne

Nodes
Speed-up
0.00 2.00 4.00 6.00 8.00 10.00 12.00 14.00
1 2 3 4 5 6 7 8 9 10 11 12

DD-MOS-F Myrinet
DD-MOS-F Ethernet
DD-MOS-F LAM
LP-MOS-F Myrinet
DLP-MOS-F Ethernet
LP-MOS-F LAM
Ideal
Performance evaluation: Parallel performance of poly-azomethyne benchmark

DD-MOS-F vs LP-MOS-F. Parallel Performance.
Poly-azomethyne. SCRe Myrinet
Performance evaluation: Communication costs

Profiling tool: Vampir 2.5, the Pallas graphical performance analysis tool for MPI applications

IA32 cluster, 8 nodes, SCore Myrinet
The distributed-data version DD-MOS-F scales better than the locally parallelized LP-MOS-F code under all tested networks.

Poor scaling of LP-MOS-F is mainly caused by the need to redistribute data locally and by the limited scalability of the divide-and-conquer eigensolver employed.

LP-MOS-F outperforms DD-MOS-F code for all benchmark jobs on any number of processors from the tested range.

Using Myrinet and Gigabit Ethernet under the SCore parallel environment provides good parallel performance.

In contrast, LAM-MPI and MPICH are not utilizing the low latency and high bandwidth of the Gigabit Ethernet network very well.
Conclusions

Two approaches to parallelizing the semiempirical code MOS-F were proposed.

Care must be taken in the choice of eigensolver to be used, as routines based on inverse iteration can scale very poorly under certain conditions.

The distributed-data parallel version of MOS-F has the advantage that memory usage and computational time scale with increasing number of processors and this will make calculations on molecules with thousands of atoms feasible.

The locally parallelized version is more efficient in computational time for small processor counts but requires more memory and scales poorly. This version would be useful for systems comprising of a moderate number of atoms.

The choice of MPI implementation is shown to have a dramatic effect on the parallel performance of MOS-F, with the "standard" LAM/MPI and MPICH performing poorly. In contrast, MPICH under the SCore parallel environment and MPICH-GM over Myrinet scale well.
Further work

- Combine the best features from LP-MOS-F and DD-MOS-F
  - Use of LAPACK and BLAS with full matrix storage on a single processor if memory permits
  - Use of ScaLAPACK with block size governed by available memory
    - For small systems, use the LP-MOS-F approach
    - For large systems, use the DD-MOS-F approach