FUJITSU THE POSSIBILITIES ARE INFINITE

Parallelization of the Molecular Orbital Program MOS-F

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

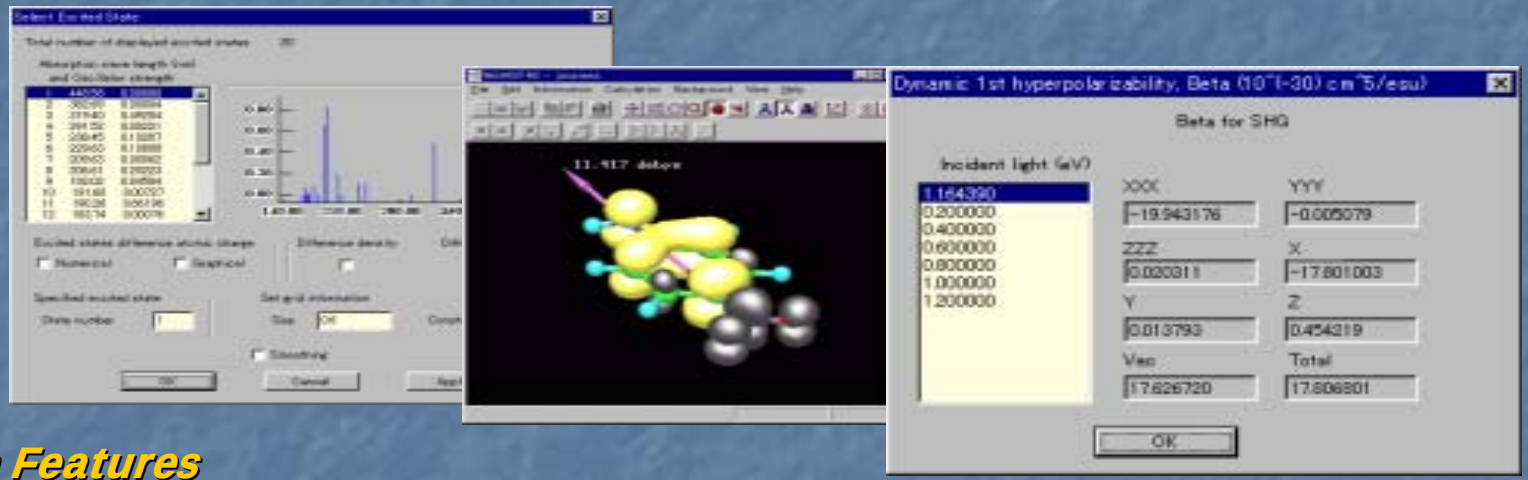
November 2003

Plan of talk

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

Introduction to MOS-F

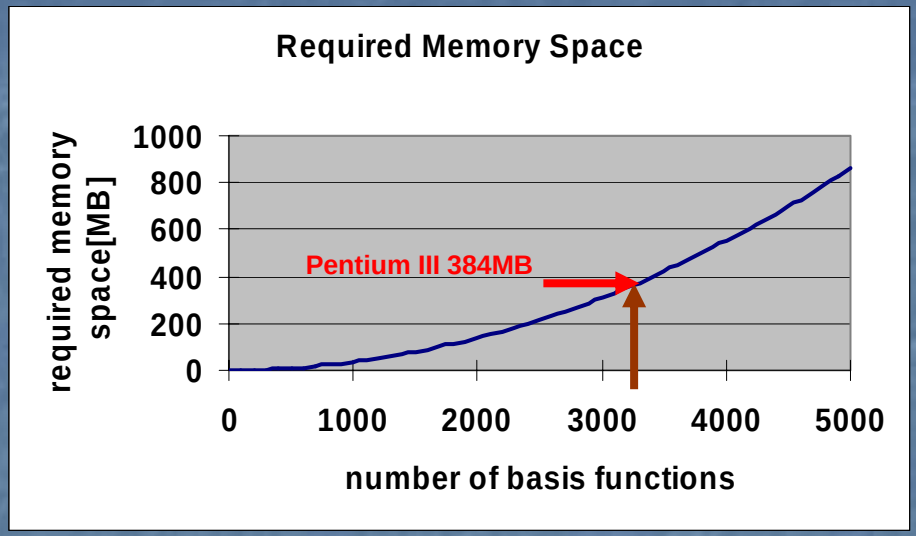
MOS–F is a semiempirical molecular orbital package for spectroscopy developed by A. Matsuura of Fujitsu Laboratories



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 (α), first hyperpolarizability (β) and second hyperpolarizability (γ)
- Coordinate input from Gaussian Z-matrix or MOPAC internal coordinates
- WinMOPAC IO interface

Introduction to MOS-F: Computational limitations



- Large memory requirements: ~ 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

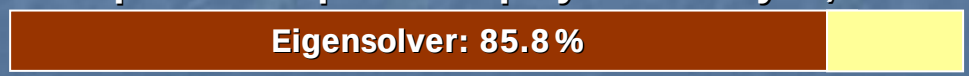
Two major components of the computational time for typical MOS-F calculations:

- Determination of all eigenvalues and eigenvectors of a real symmetric matrix
- Matrix multiplication

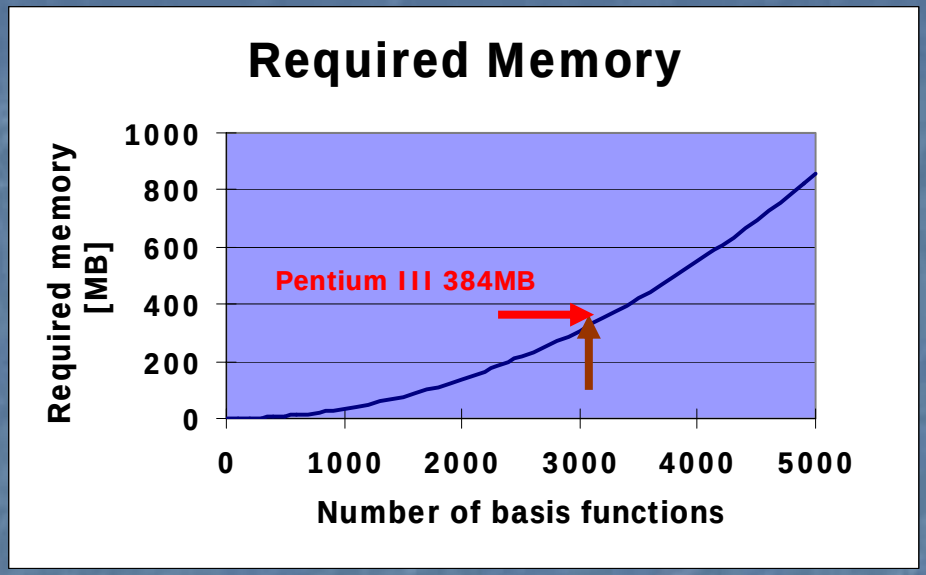
To perform MOS-F calculations efficiently:
Optimized eigensolver and matrix multiplication routines are required

Computational profile of crambin, n=1,622

Computational profile of poly-azomethyne, n=807



Introduction to MOS-F: Computational limitations



- Large memory requirements: ~ 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

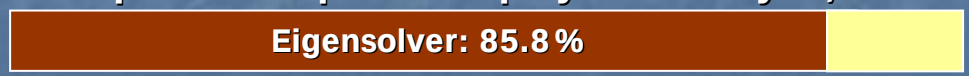
Two major components of the computational time for typical MOS-F calculations:

- 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

Computational profile of poly-azomethyne, n=807



Parallelization approaches

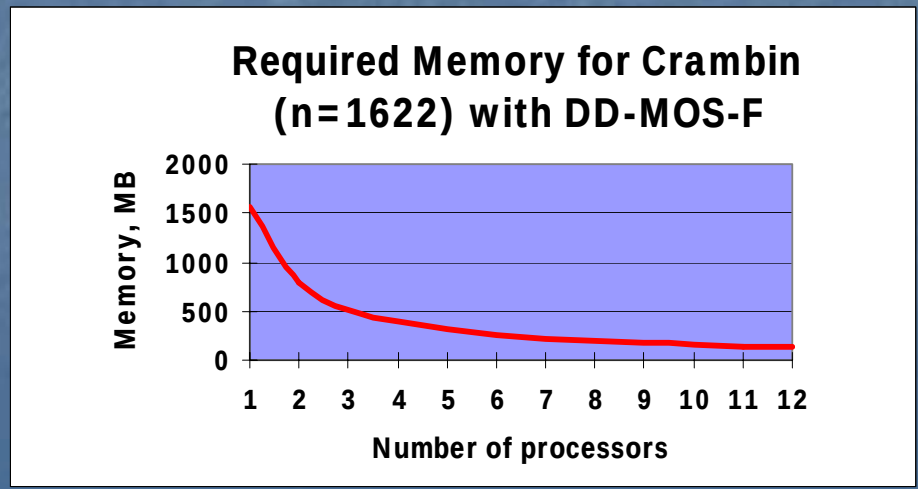
Two different approaches to parallelizing MOS-F:

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



Choice of eigensolver: LAPACK

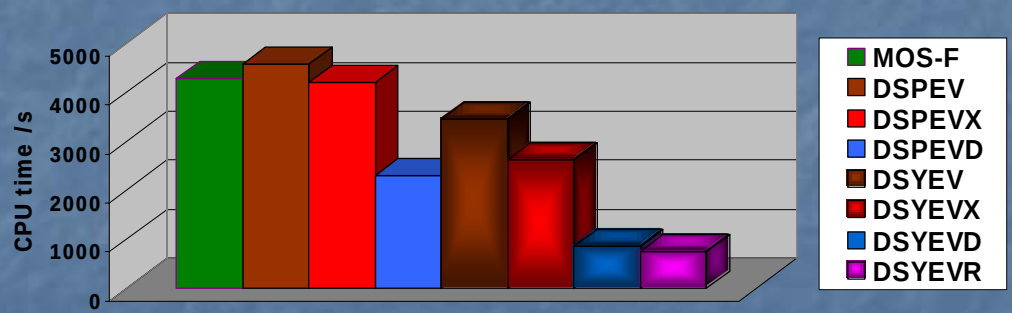
Eigensolvers used in the testing procedure

Name	Matrix storage	Algorithm
MOS-F eigensolver (EISPACK)	Packed	Rational QL + inverse iteration
DSY(P)EV	Full (Packed)	Implicitly shifted QR
DSY(P)EVX	Full (Packed)	Implicitly shifted QR or bisection + inverse iteration
DSY(P)EVD	Full (Packed)	Divide and conquer (DC)
DSYEVR	Full	Relatively robust representations (RRR)

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

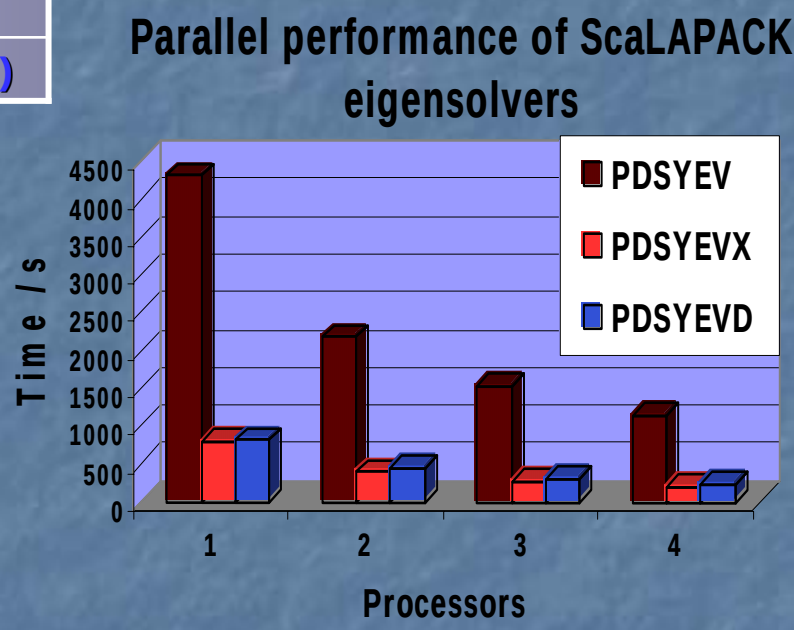
Choice of eigensolver: ScaLAPACK

Parallel eigensolvers used in the testing procedure

Name	Matrix storage	Algorithm
PDSYEV	**Full**	**Implicitly shifted QR**
PDSYEVX	**Full**	**Bisection + inverse iteration**
PDSYEVD	**Full**	**Divide and conquer (DC)**

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

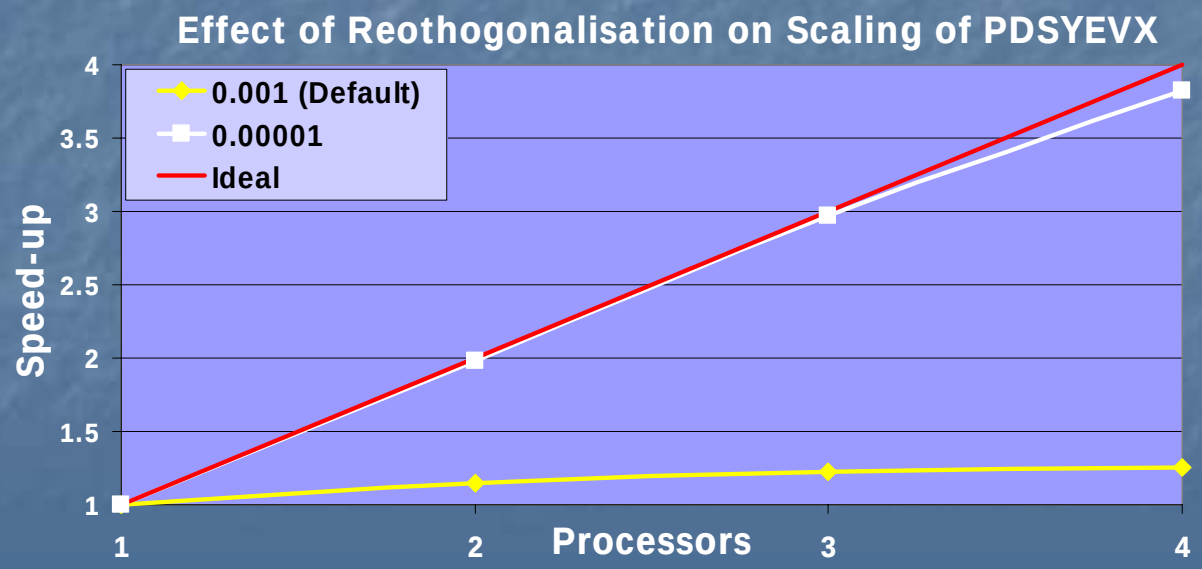
There is little to choose between PDSYEVX and PDSYEVD on the basis of wall time performance



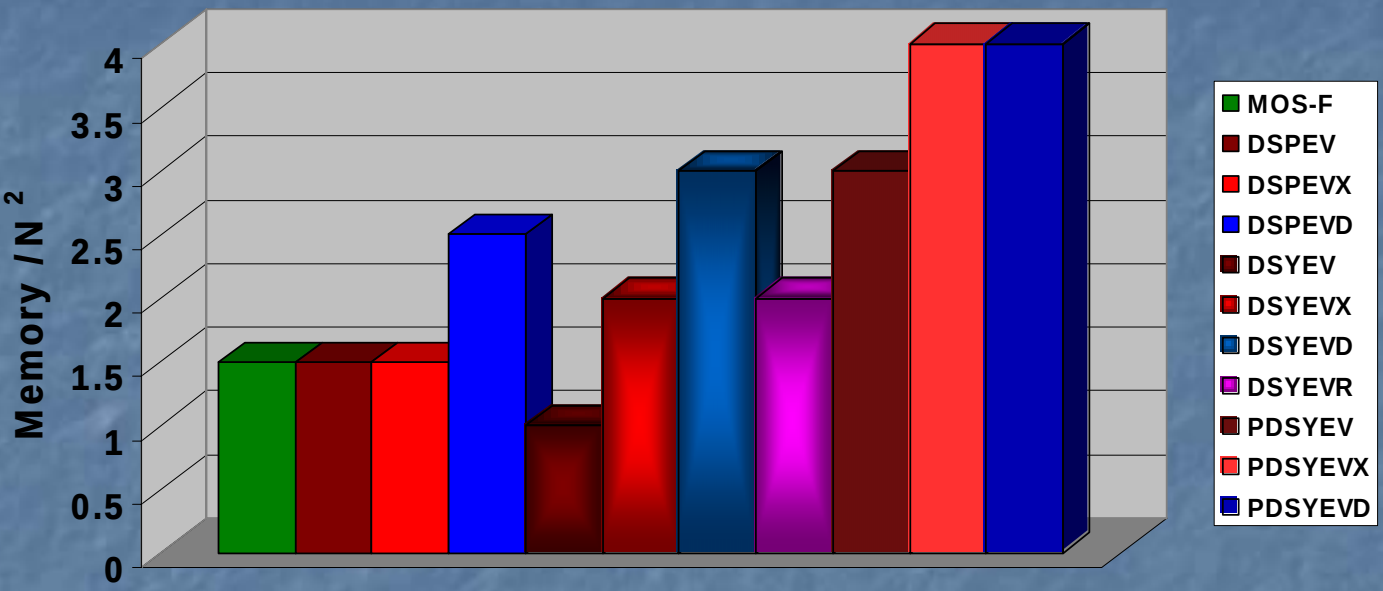
Input Matrix: 4225 x 4225
Machine environment: A cluster of four dual-processor 733 MHz Pentium III nodes connected via Myrinet

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"*



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

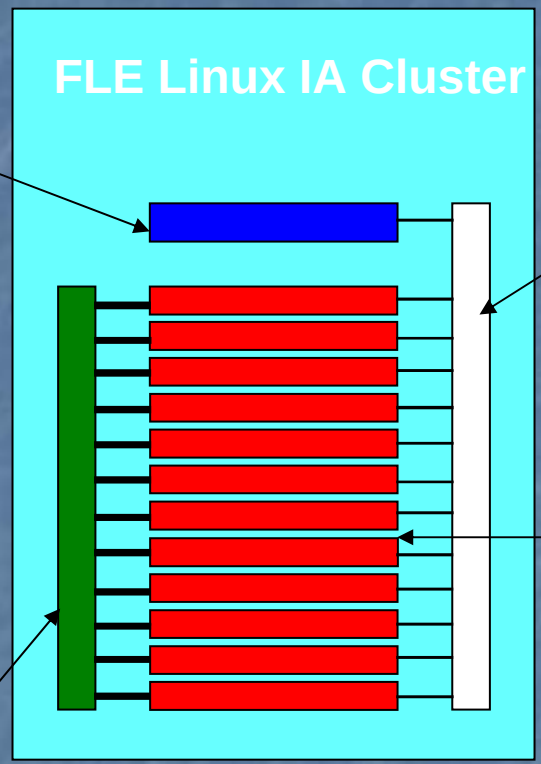
Parallelization approaches: Key features

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

- 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

Front-end server
Dual 2.4 GHz Pentium Xeon, 1 GB memory, 120 GB disk
SCore parallel environment
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

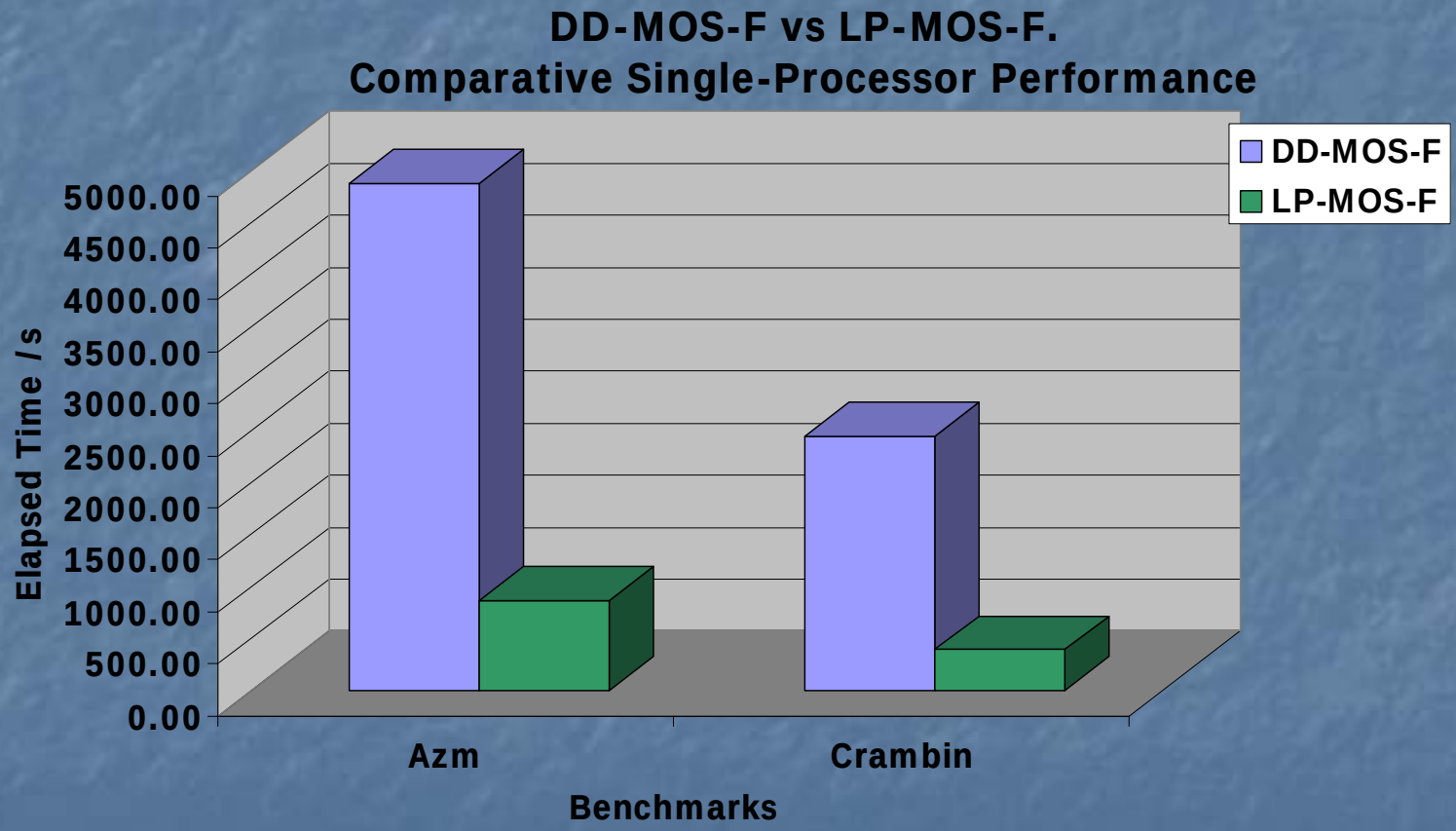
Performance evaluation: Benchmarks

Job	Formula	Number of Atoms	Number of Basis Functions	Method
crambin	$C_{202}H_{314}N_{55}O_{64}S_6$	641	1622	CNDO/S SCF

Job	Formula	Number of Atoms	Number of Basis Functions	Method
Poly-azomethyne (azm)	$C_{153}H_{111}N_{21}$	285	807	CNDO/S CI(80,80)

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



Compiler: pgf90 using *–fast –tp p7 –Mdalign* optimization
LP-MOS-F uses **LAPACK 3.0** and **Intel MKL 5.2** for BLAS

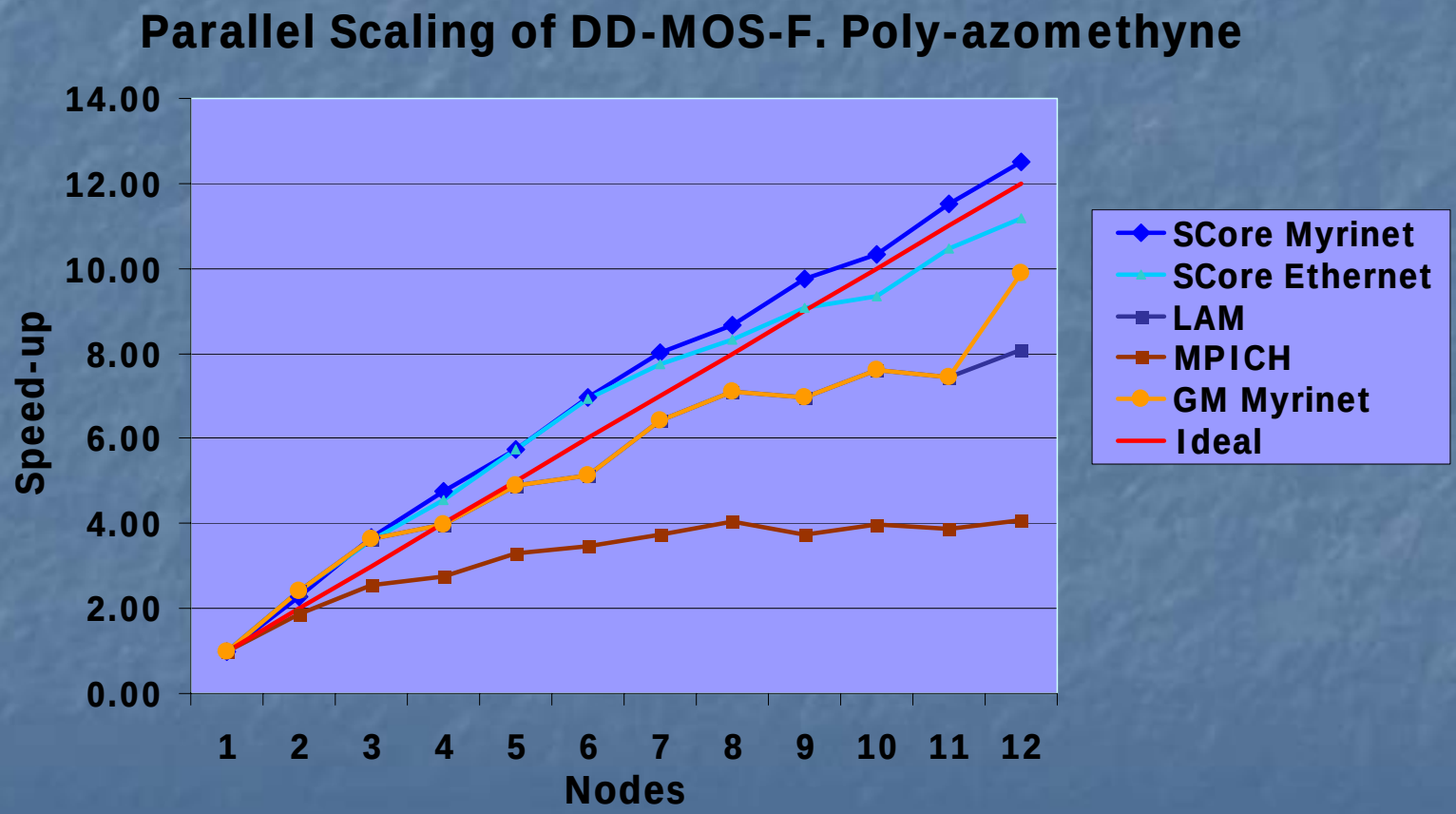
Performance evaluation: Parallel performance

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

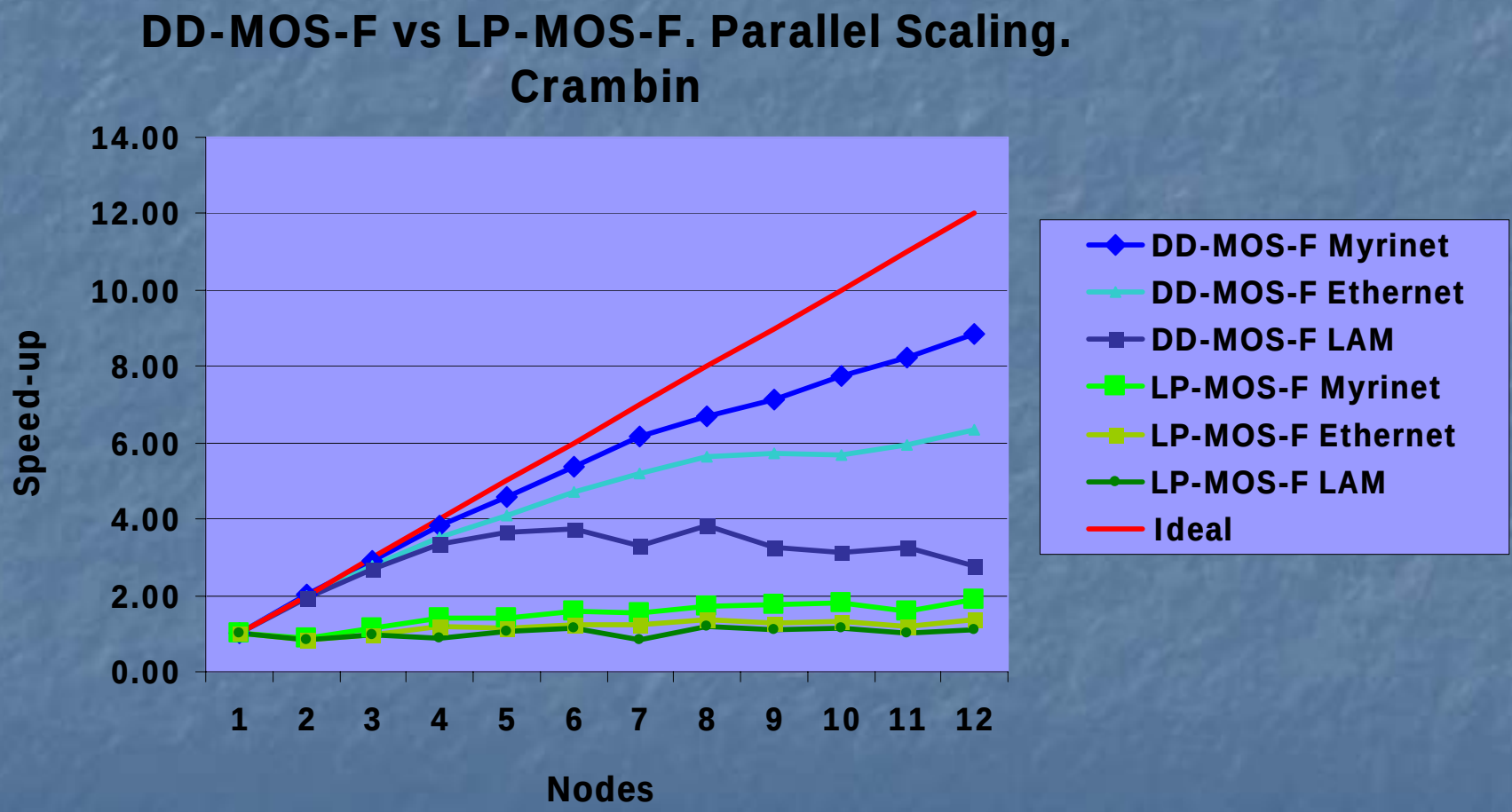
- **LAM 6.5.9**
- **MPICH 1.2.4**
- **MPICH 1.2.4** implemented under **SCore 5.4.0** using **Gigabit Ethernet** network
- **MPICH 1.2.4** implemented under **SCore 5.4.0** using **Myrinet 2000** network
- **MPICH-GM 1.2.4..8a** under **GM 1.6**

PGI compilers (pgf77, pgf90) with *-fast –tp p7 –Mdalign* optimization were used in all cases

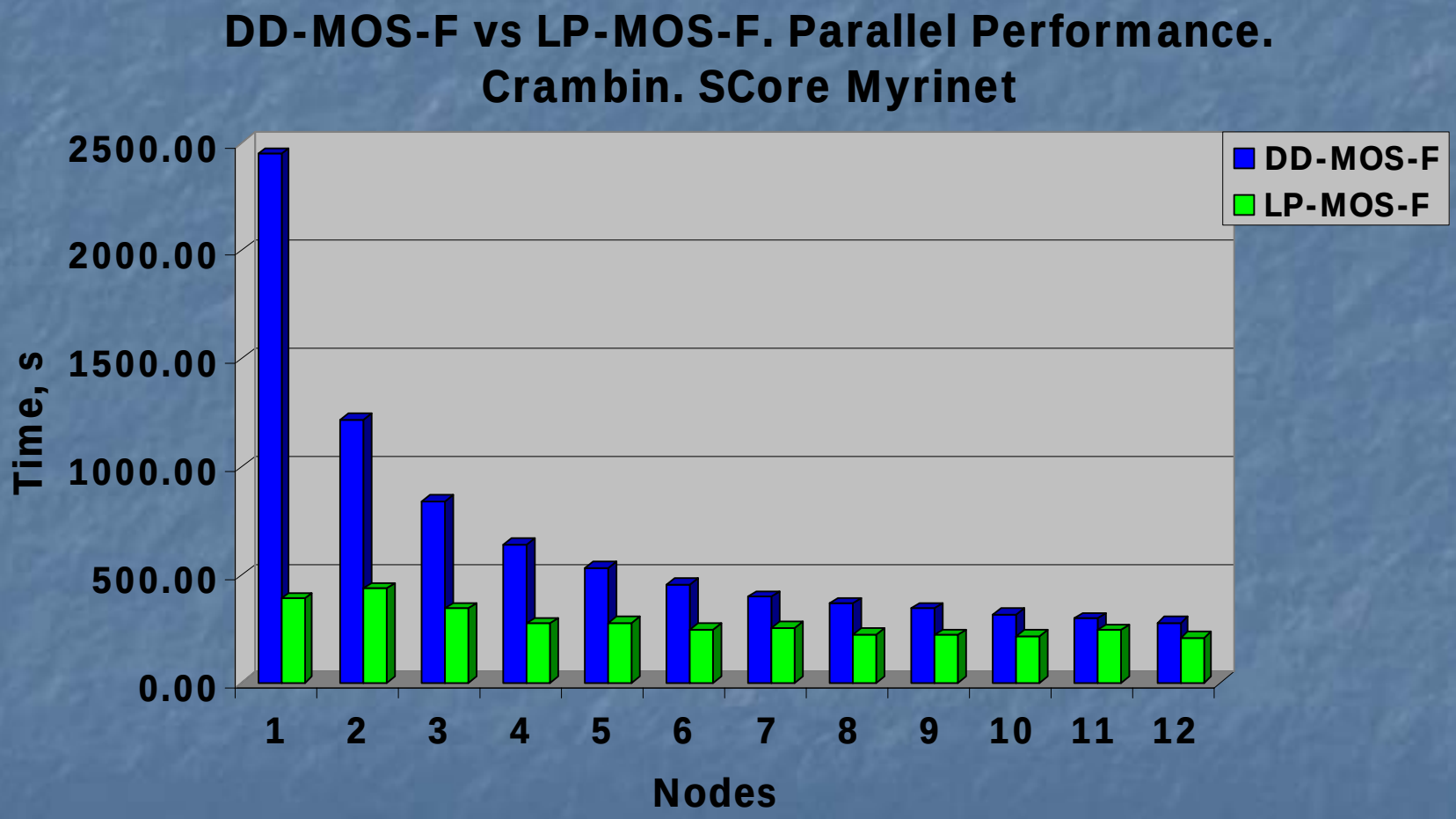
Performance evaluation using various MPI implementations



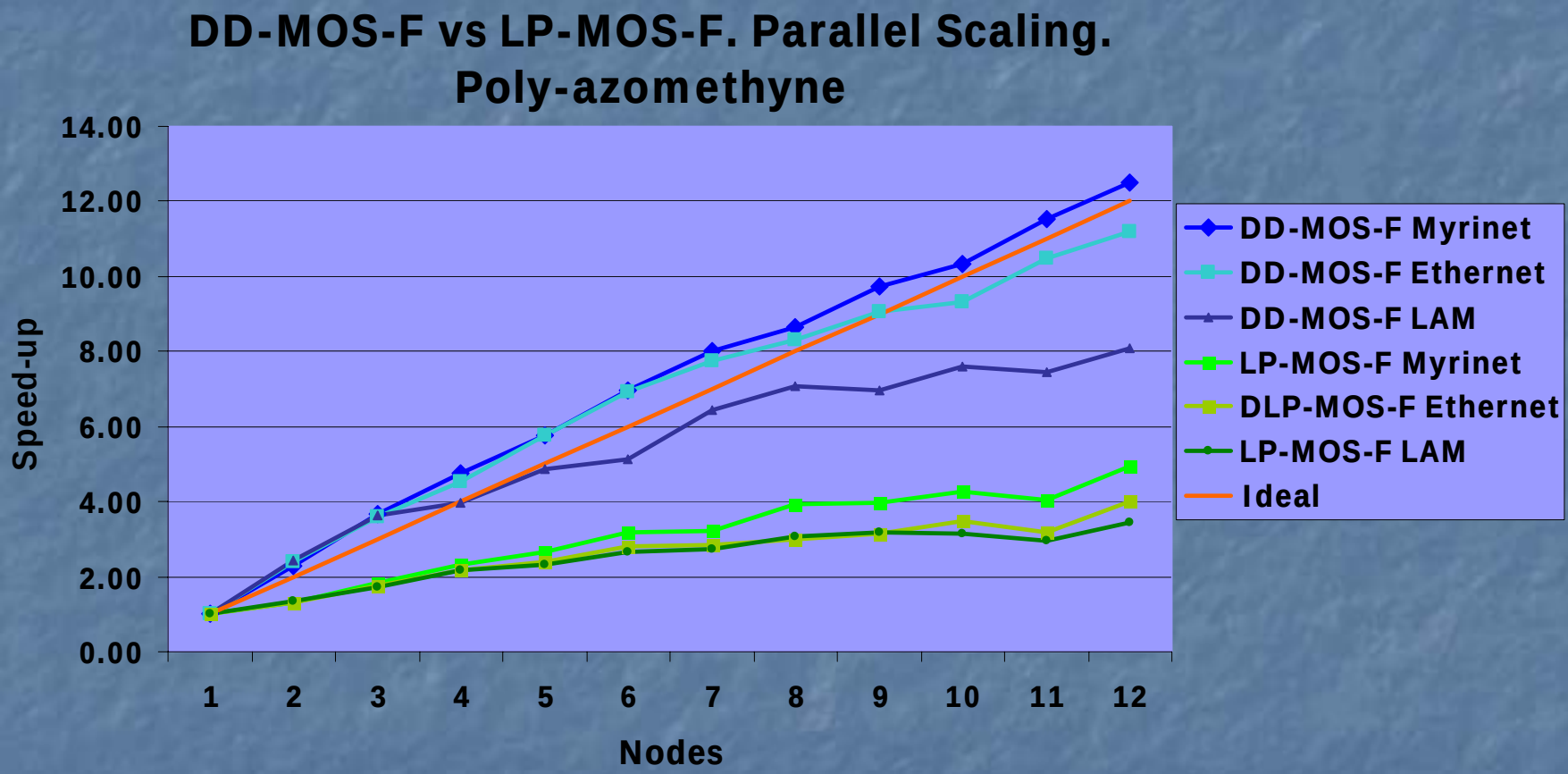
Performance evaluation: Parallel scaling of crambin benchmark



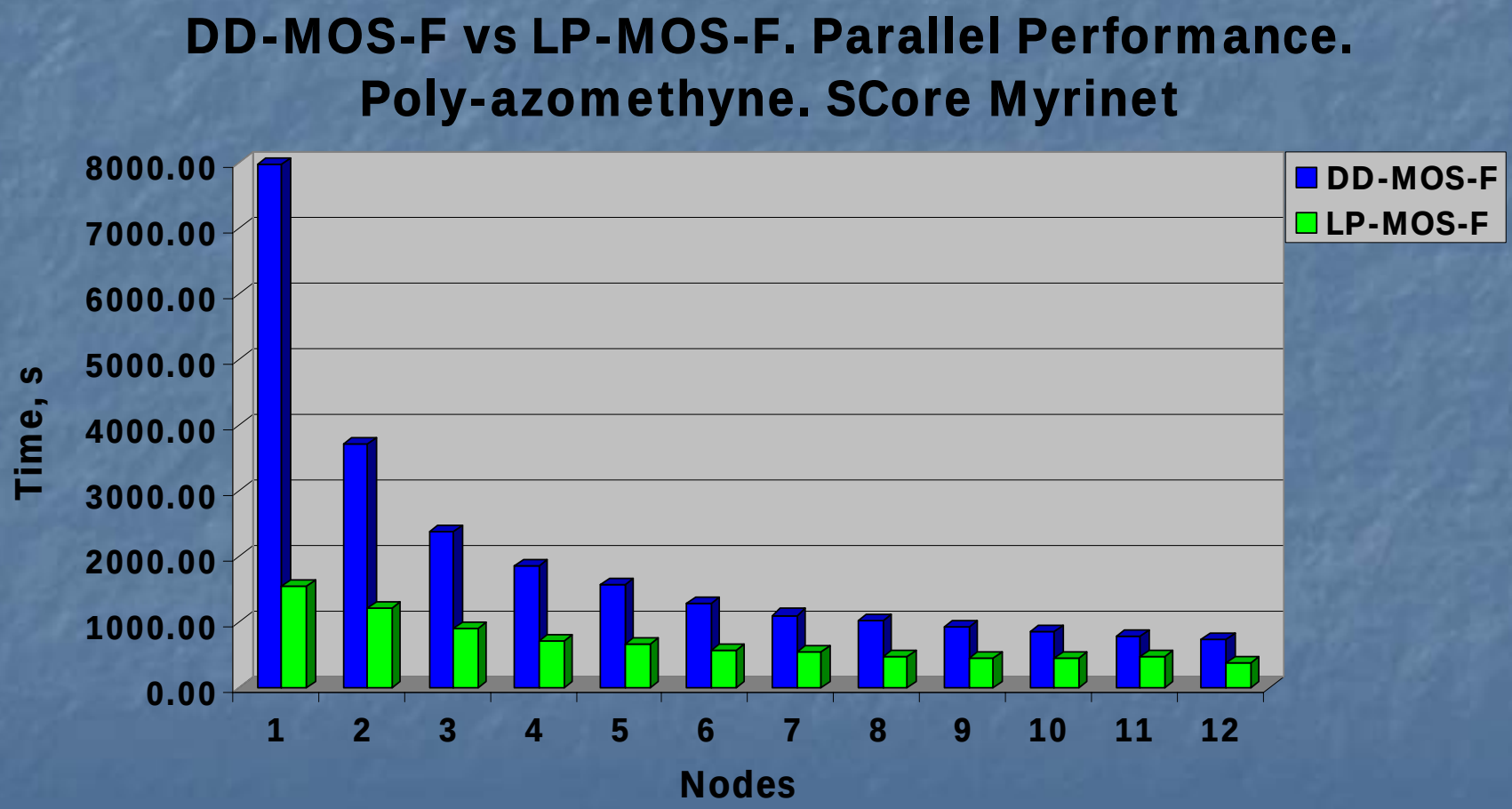
Performance evaluation: Parallel performance of crambin benchmark



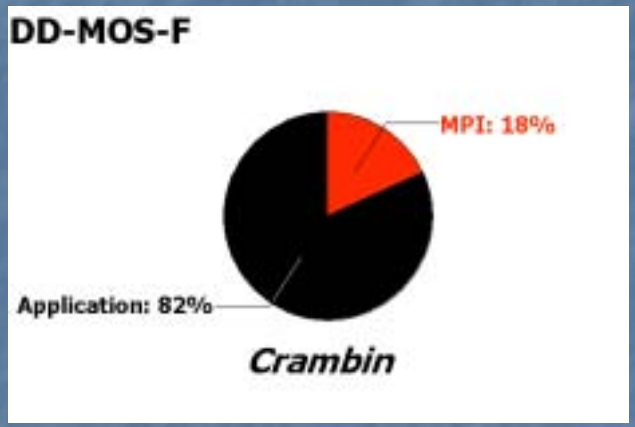
Performance evaluation: Parallel scaling of poly-azomethyne benchmark



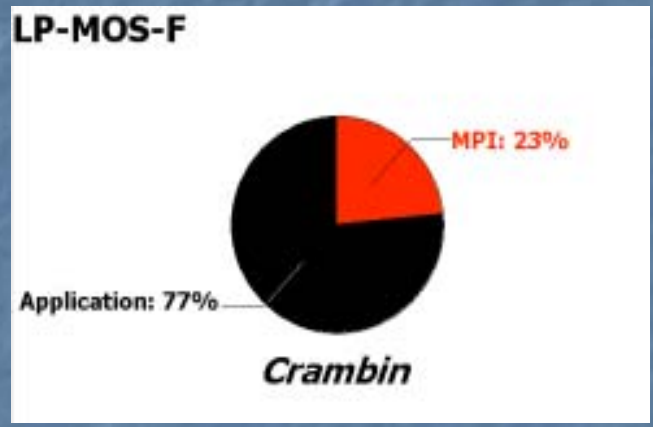
Performance evaluation: Parallel performance of poly-azomethyne benchmark

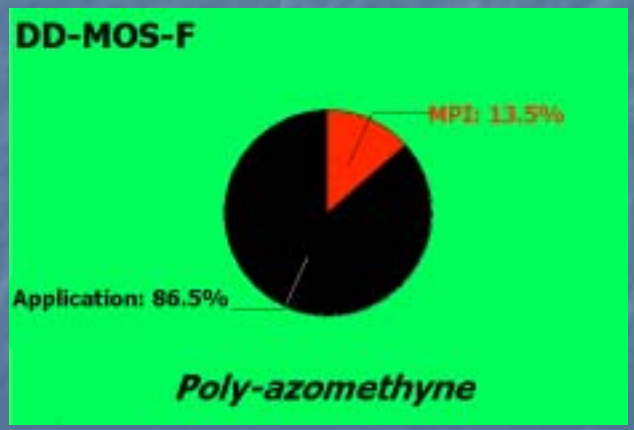


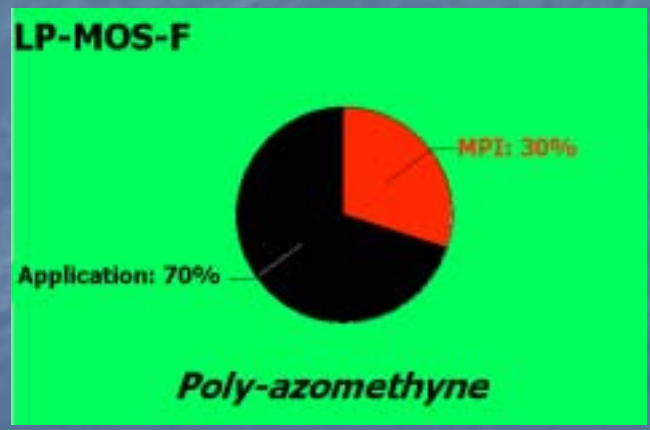
Performance evaluation: Communication costs



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







IA32 cluster, 8 nodes, SCore Myrinet

Performance evaluation: Summary

- 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