ACESIII: Parallel implementation of coupled-cluster methods, a practical perspective

- Design philosophy
- Basics of SIAL (Super Instruction Architecture Language)
- Applications/Method implemented

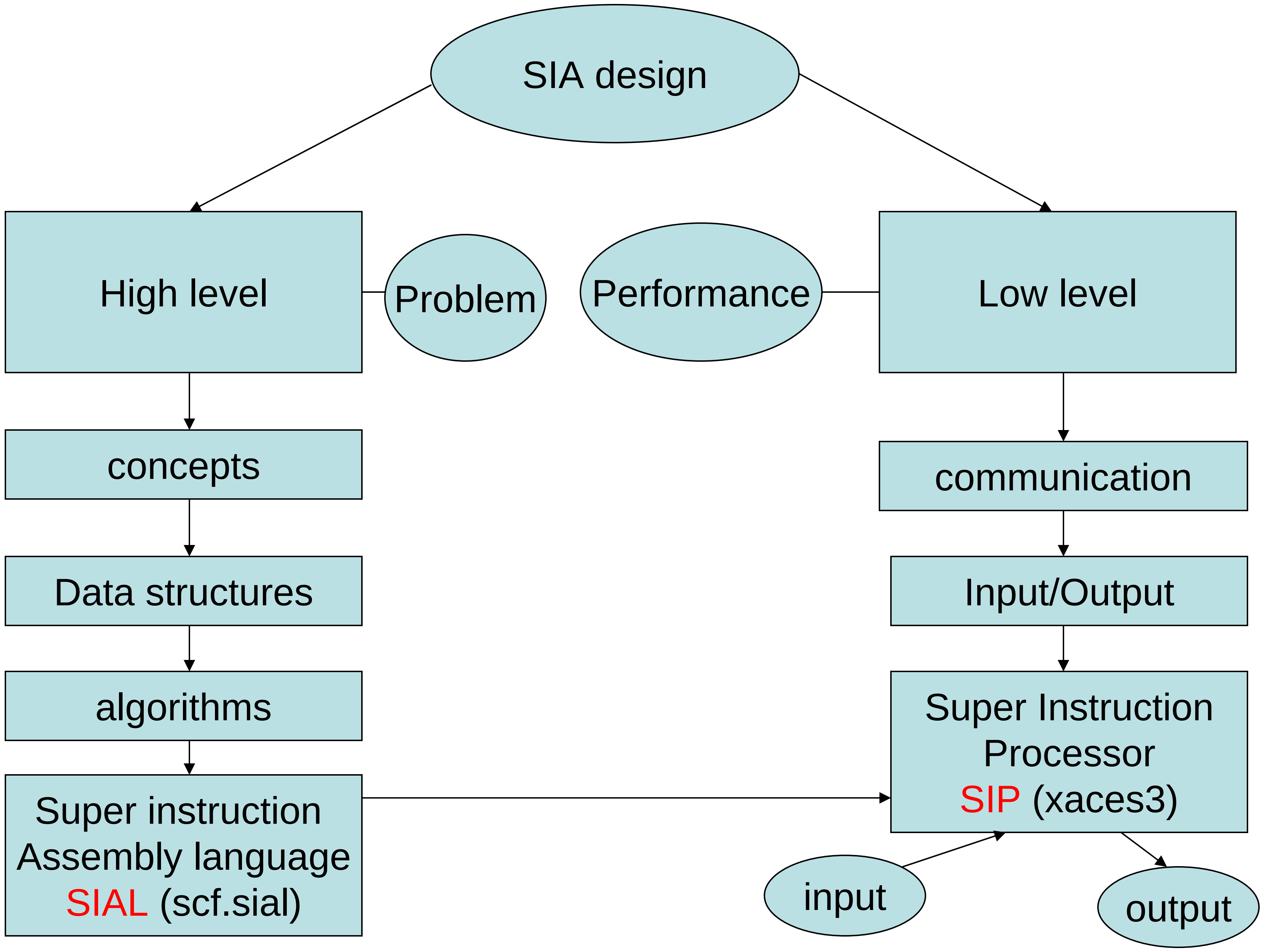
  - small selection
- Conclusions/Improvements

Design Philosophy

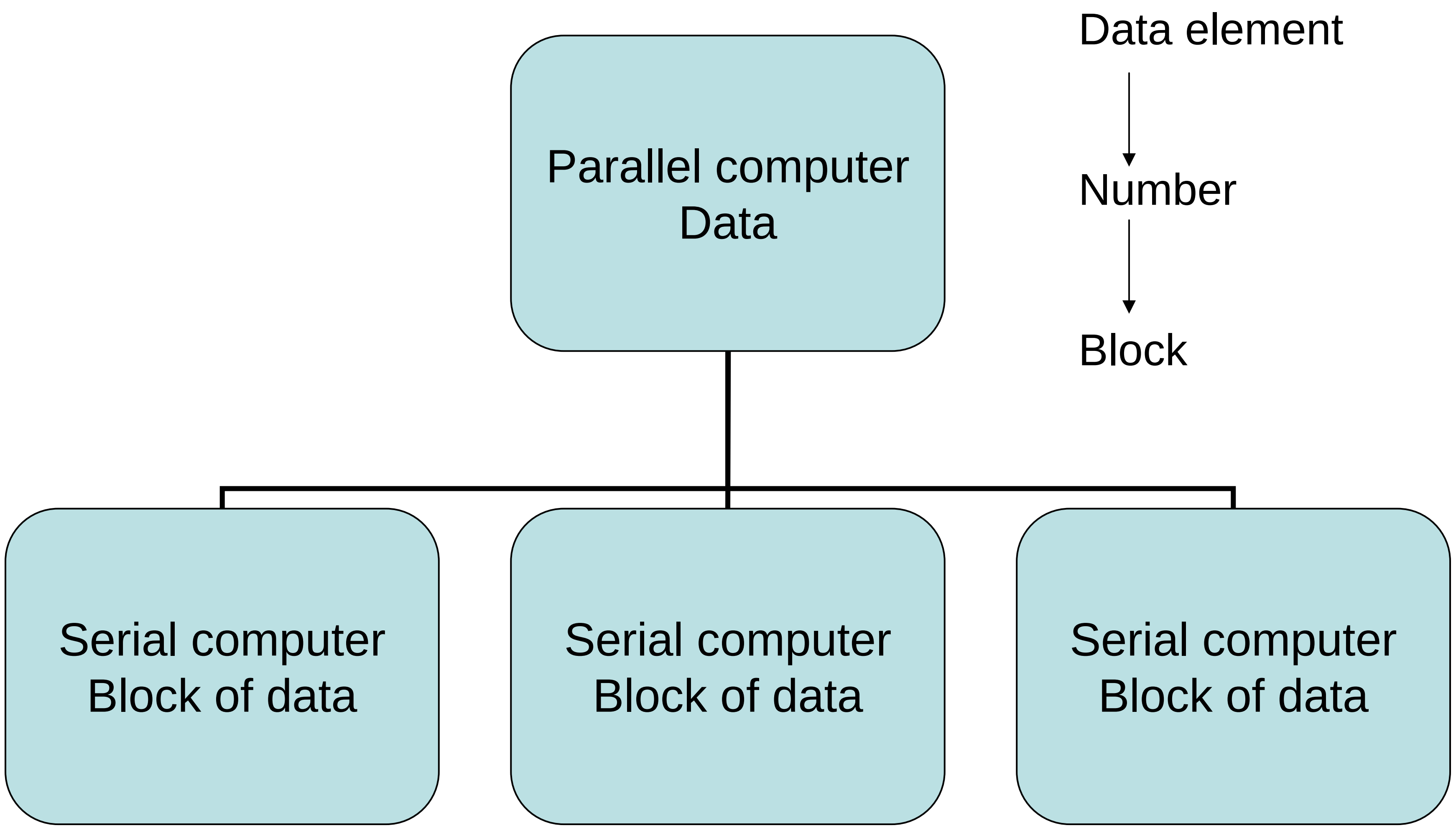
- Two fundamental principles

  1. Parallel Computer $\Longleftrightarrow$ ‘super’ serial computer

  2. Program execution is separated from the application specific algorithmic design


SIA design
High level
Problem
Performance
Low level
concepts
Data structures
algorithms
Super instruction Assembly language SIAL (scf.sial)
communication
Input/Output
Super Instruction Processor SIP (xaces3)
input
output

Parallel Computer



Advantages

- Block operations take time(dependent on block size -> tunable)
- Message ‘hiding’ possible
- Flexibility in scheduling tasks

Simplicity: Algorithm/Execution separation

- Design a ‘simple’ language to express the algorithm (SIAL)

- Details of execution determined at a lower level

- A precise boundary exists

Benefits

- More efficient use of expertise
- More efficient tuning
- ***Easier extension to other disciplines***

SIAL: THE BASICS

- Principles

  - Index segmentation

  - Blocking of arrays

  - Parallelization

  - Anomalies

- Specific

  - Standard operations
    - matrix mult
    - addition
    - ect…

  - ‘Non standard’ operations
    - computing integrals

Segmentation

- The range of each index is divided into a relatively small number of segments which are determined by defining a segment size.
- There can be different segment sizes for different types of indices.

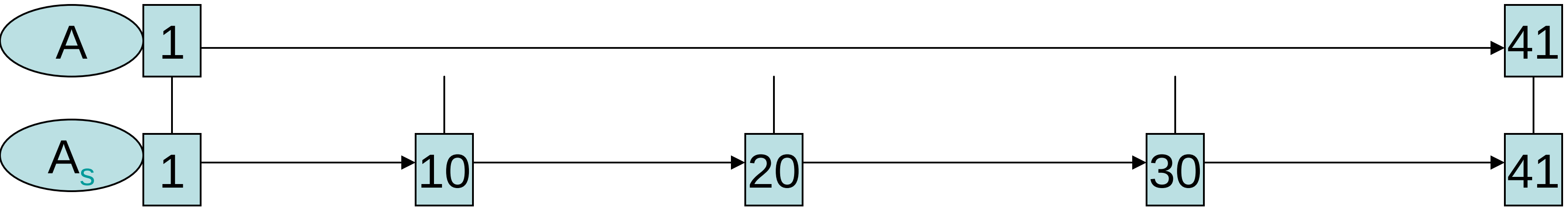
BLOCKS

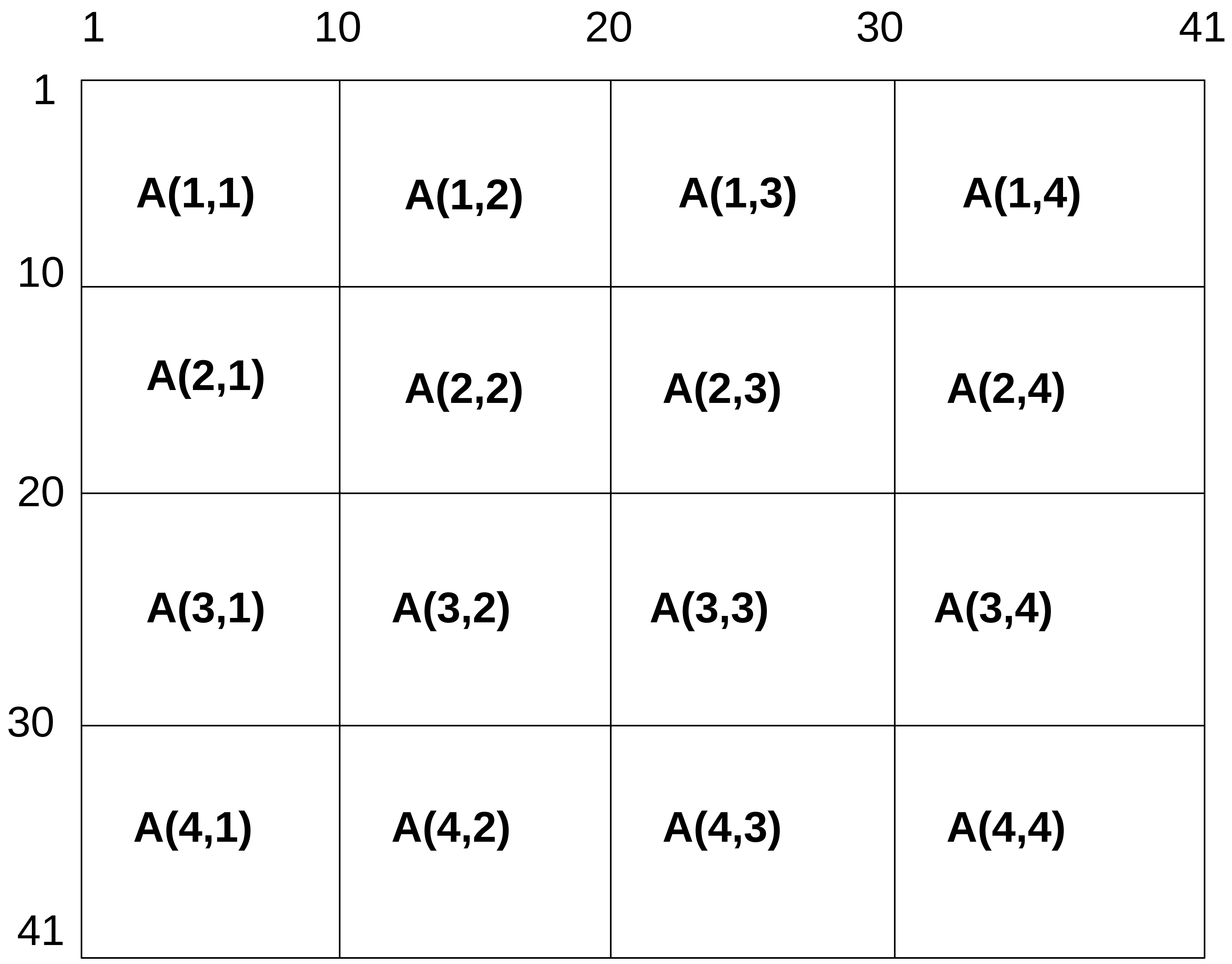
- Arrays are decomposed into blocks the size of which is determined by the index segments.
- Blocks are the basic entities which operations are performed on/with.
- Blocks should be small so that many of them fit into memory.

- Array $\longrightarrow$ • Array(Blocked)
- A(n,n) $\longrightarrow$ • $A_s(N,N)$ = **A(N,N)**
- n=41 $\longrightarrow$ • N=4

Number of segments = 4

Number of blocks = 16




1
10
20
30
41
1
10
20
30
41
A(1,1)
A(1,2)
A(1,3)
A(1,4)
A(2,1)
A(2,2)
A(2,3)
A(2,4)
A(3,1)
A(3,2)
A(3,3)
A(3,4)
A(4,1)
A(4,2)
A(4,3)
A(4,4)

Arrays

- Static
- Local
- ***Temp***
- Distributed
- Served

- Replicated on each processor
- Partially replicated
- Only exists within the scope it is used
- Exists in its entirety in distributed memory
- Exists on the disk

Parallelization

- The main feature is the PARDO which determines how the work is to be distributed among the processors.
- ‘Horizontal’ load balancing: If the work is not evenly distributed keeps all processors busy.
- ‘Vertical’ load balancing: Allows multiple PARDO loops to be executed simultaneously.

Example

- $X(a,b,i,j) = \sum_{c,d} V(a,b,c,d)*T(c,d,i,j)$ indeces
- $\mathbf{X}(A,B,I,J) = \sum_{C,D} \mathbf{V}(A,B,C,D)*\mathbf{T}(C,D,I,J)$ Block

Parallelization

```
PARDO A, B, C, D
    REQUEST V(A,B,C,D)
    DO I
    DO J
        REQUEST T(C,D,I,J)
        X(A,B,I,J) = V(A,B,C,D)*T(C,D,I,J)
```

Anomalous behavior

- The self consistent field method(SCF)
  - most efficient using specially designed super instructions
  - Fock build implemented
- CCSD(T) run on > 30,000 processors
  - Uses specialized parallelization techniques not used elsewhere

Applications: Range of methods implemented

- Rank method according to
  1. Computational cost (Scaling)
  2. Data requirements
  3. Communication requirements
     a) input(reading)
     b) output(writing)
  4. Scale of 1-4 (small – large)

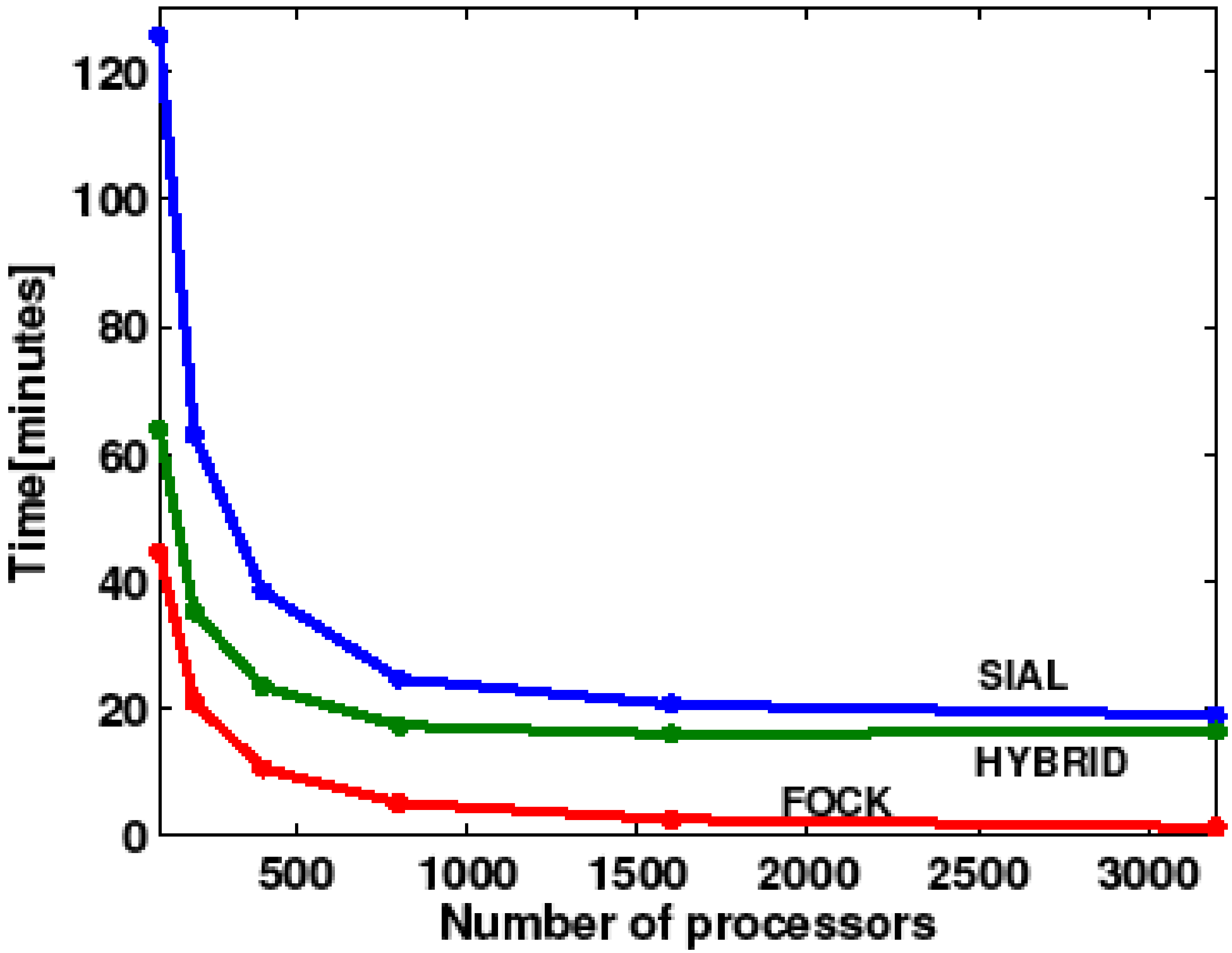
Implemented methods shown

Method	Scaling	Data	Comm.
SCF	1(N^4)	1	2
MP2 gradient	2(N^5)	2	3
CCSD	3(N^6)	4	4
CCSD(T)	4(N^7)	3	1

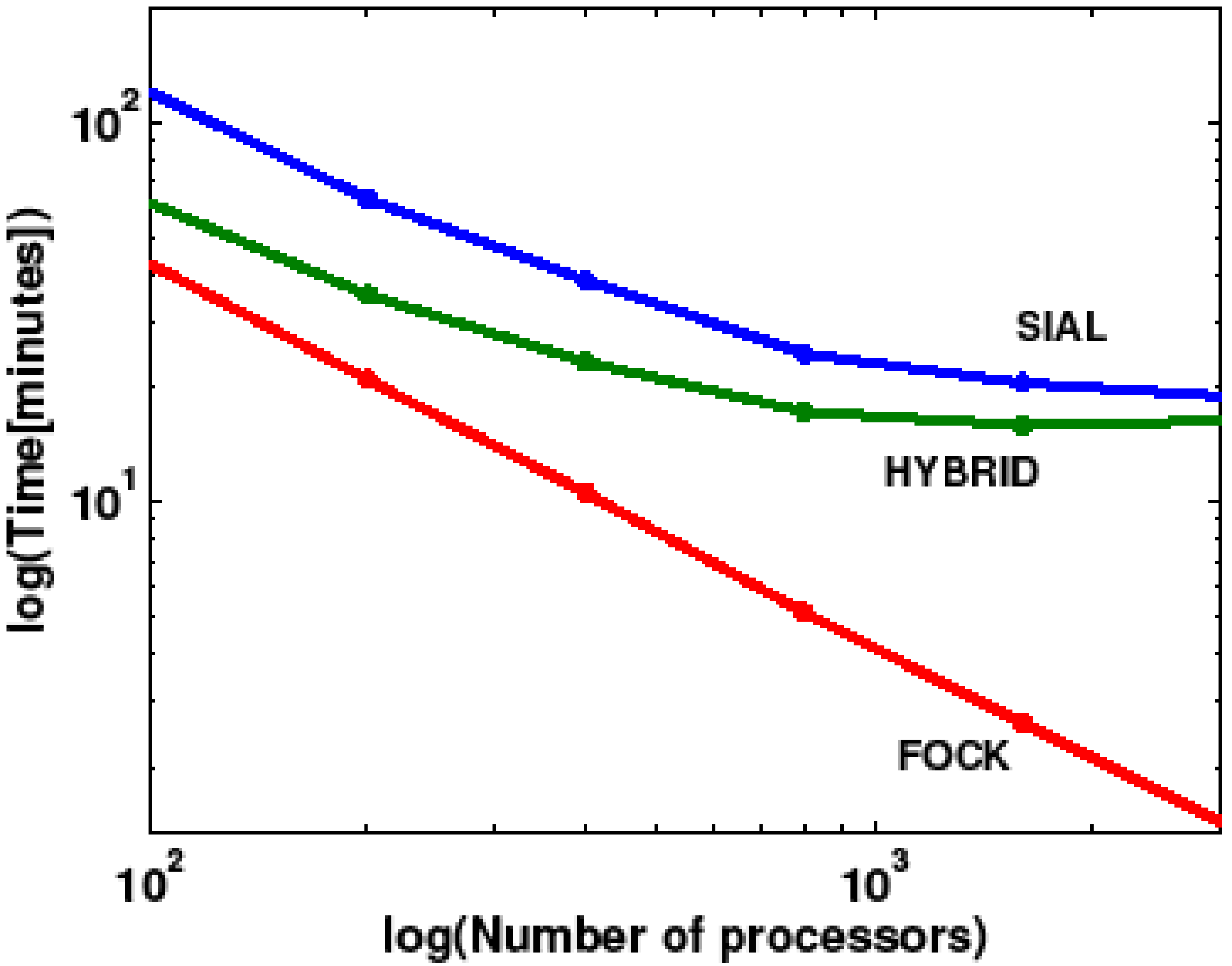
Computational Details

Method	SCF(UHF)
Molecule	RDX
Number of basis functions	1005
Number of electrons	114
Number of atoms	21

SCF(UHF) scaling results for the RDX molecule.



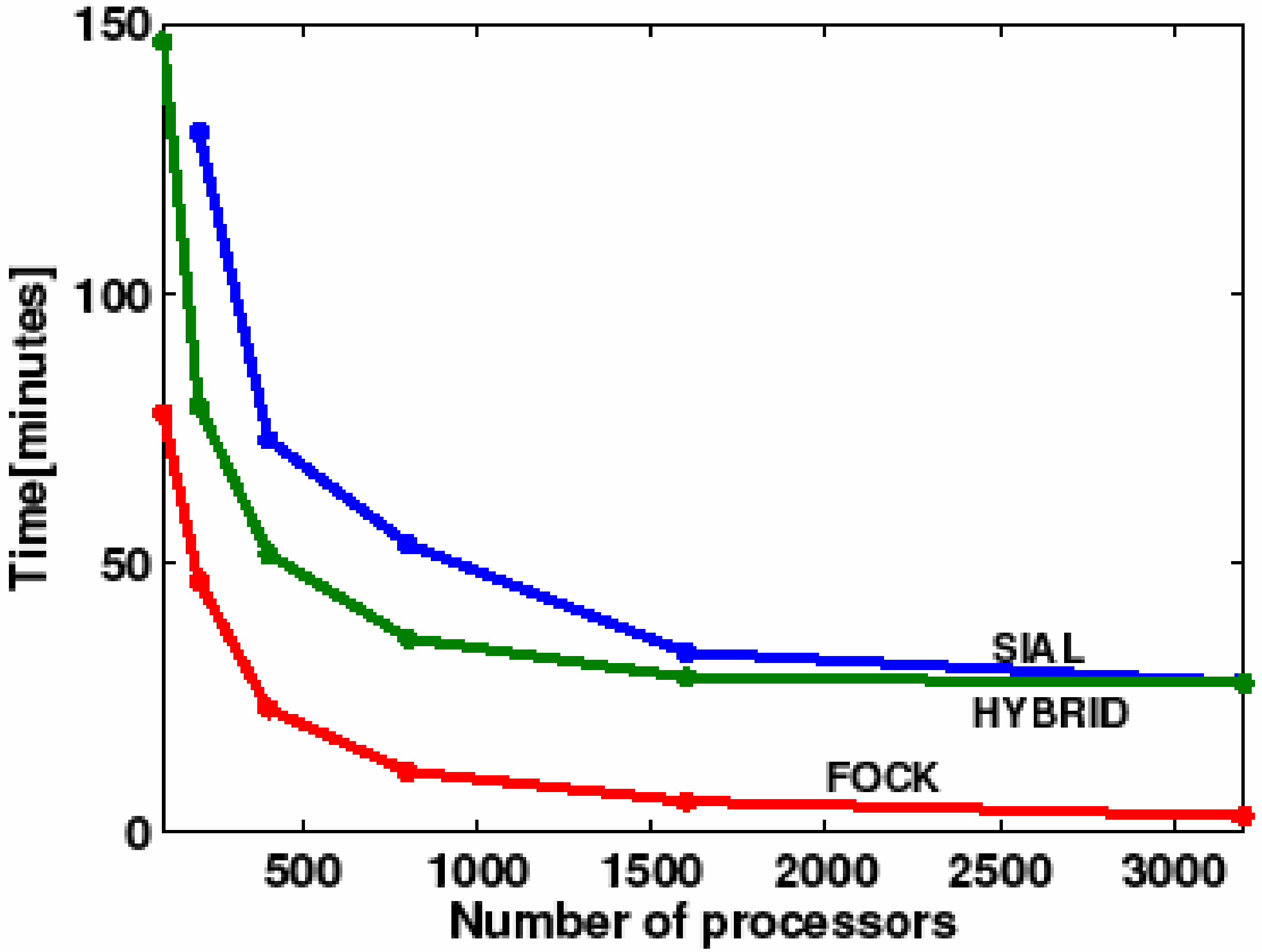
SCF(UHF) scaling results for the RDX molecule.



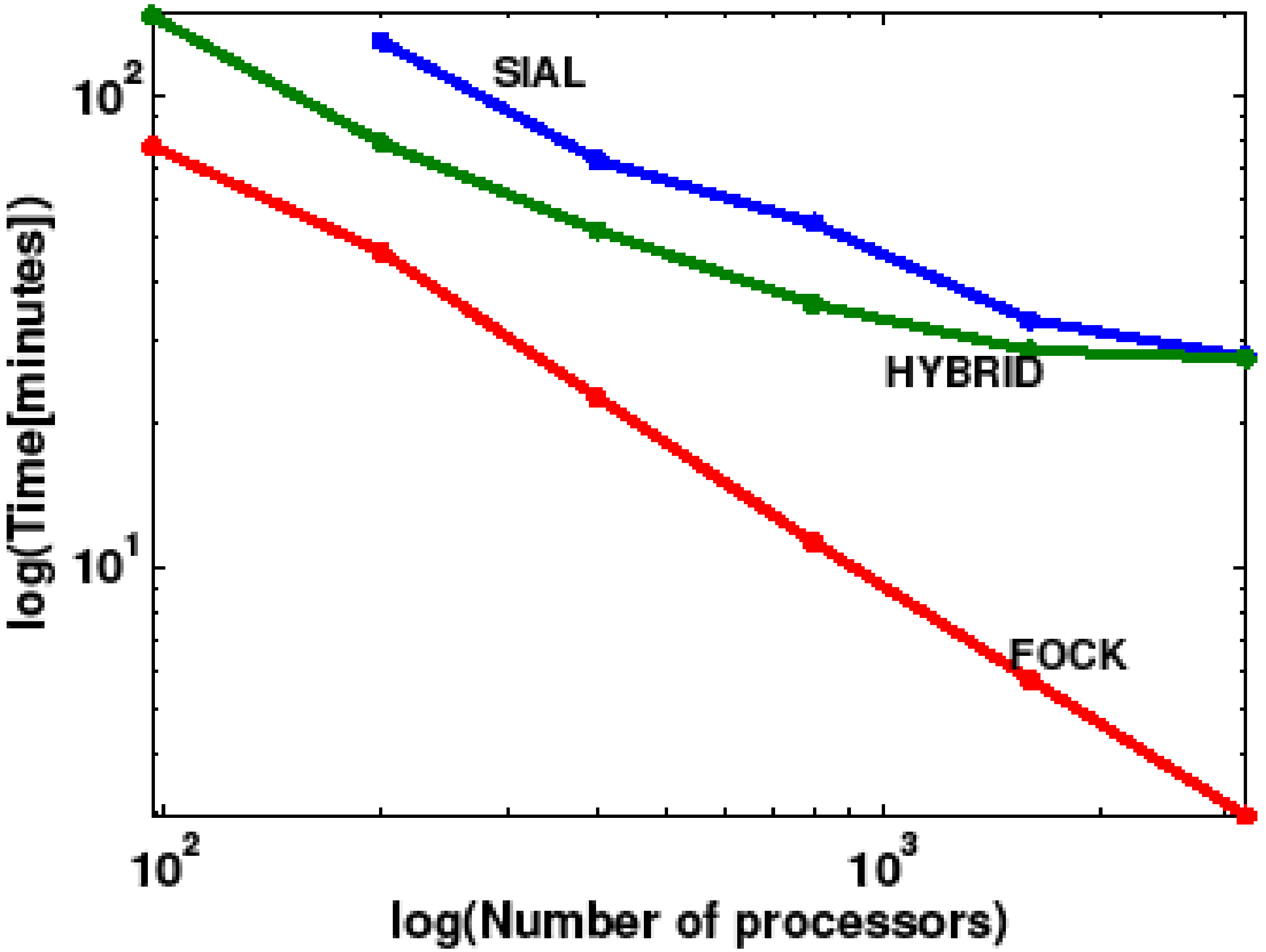
Computational Details

Method	SCF(UHF)
Molecule	$(H_20)_{21}H^+$
Number of basis functions	1232
Number of electrons	210
Number of atoms	64

SCF(UHF) scaling results for the $(H_20)_{21}H^+$ molecule.



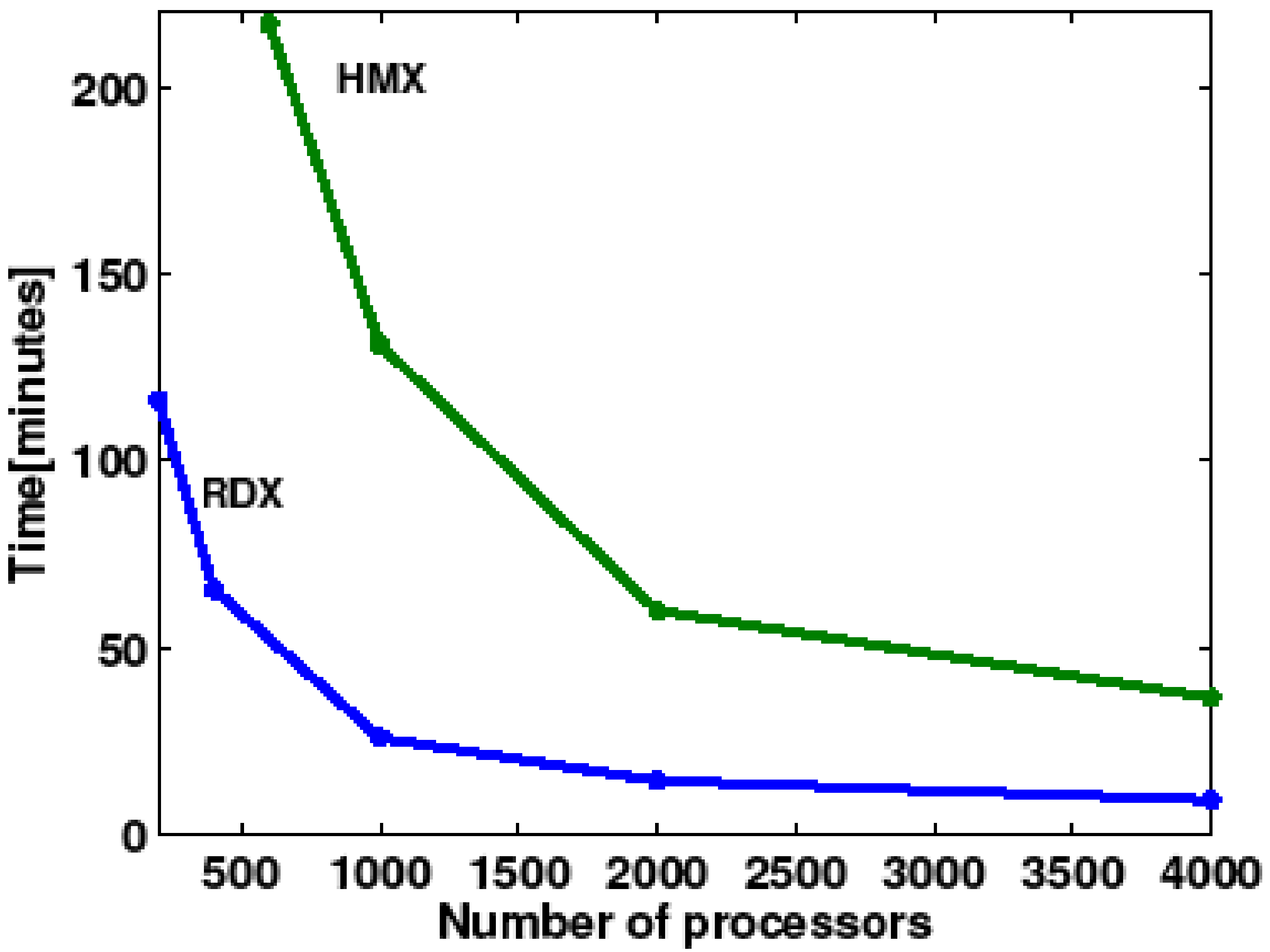
SCF(UHF) scaling results for the $(H_20)_{21}H^+$ molecule.



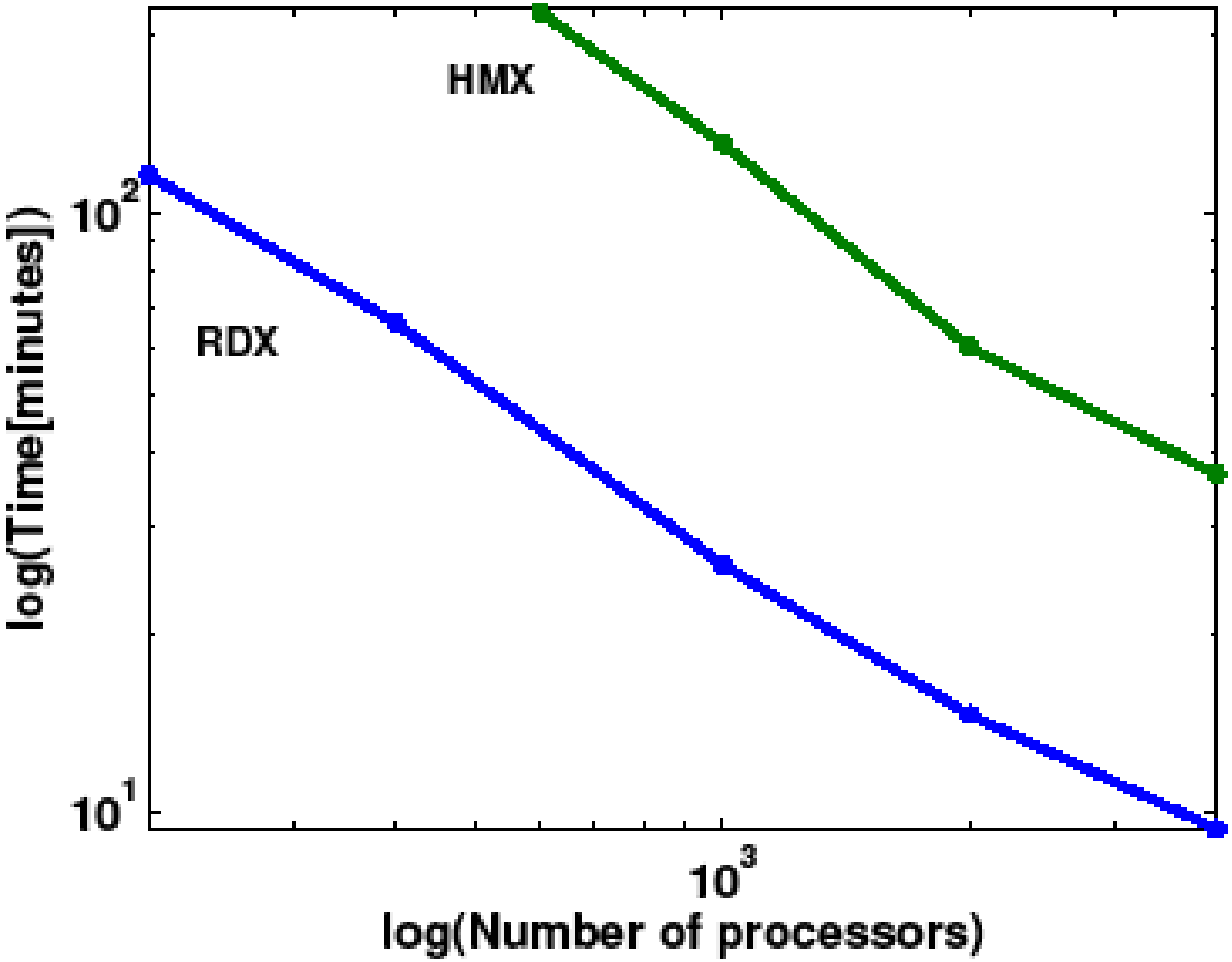
Computational details of MP2 gradient computation

Method	MP2 gradient	MP2 gradient
Molecule	RDX	HMX
Number of bf’s	1005	1340
Number of electrons	114	152
Number of atoms	21	28

MP2 gradient(RHF) scaling results.

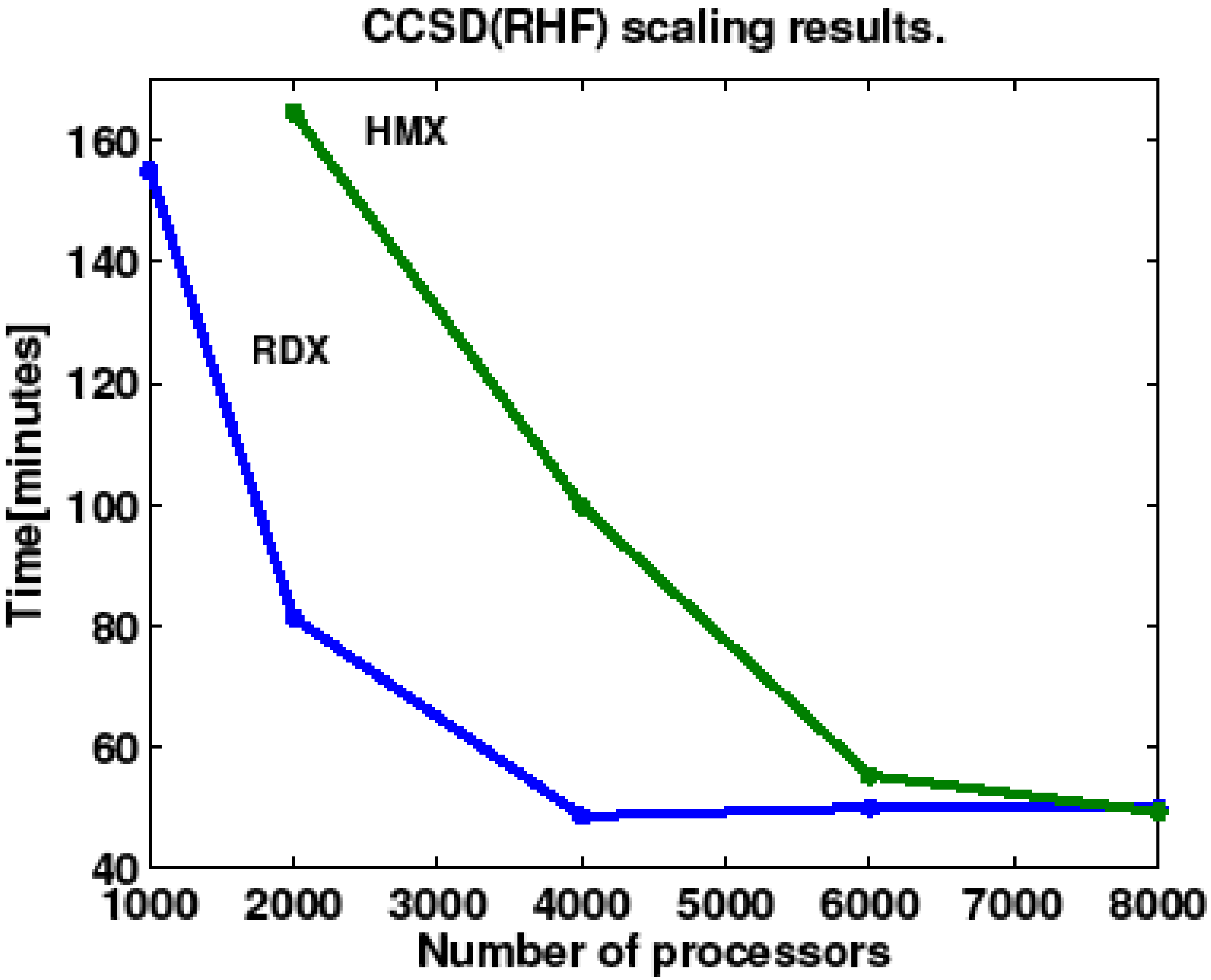


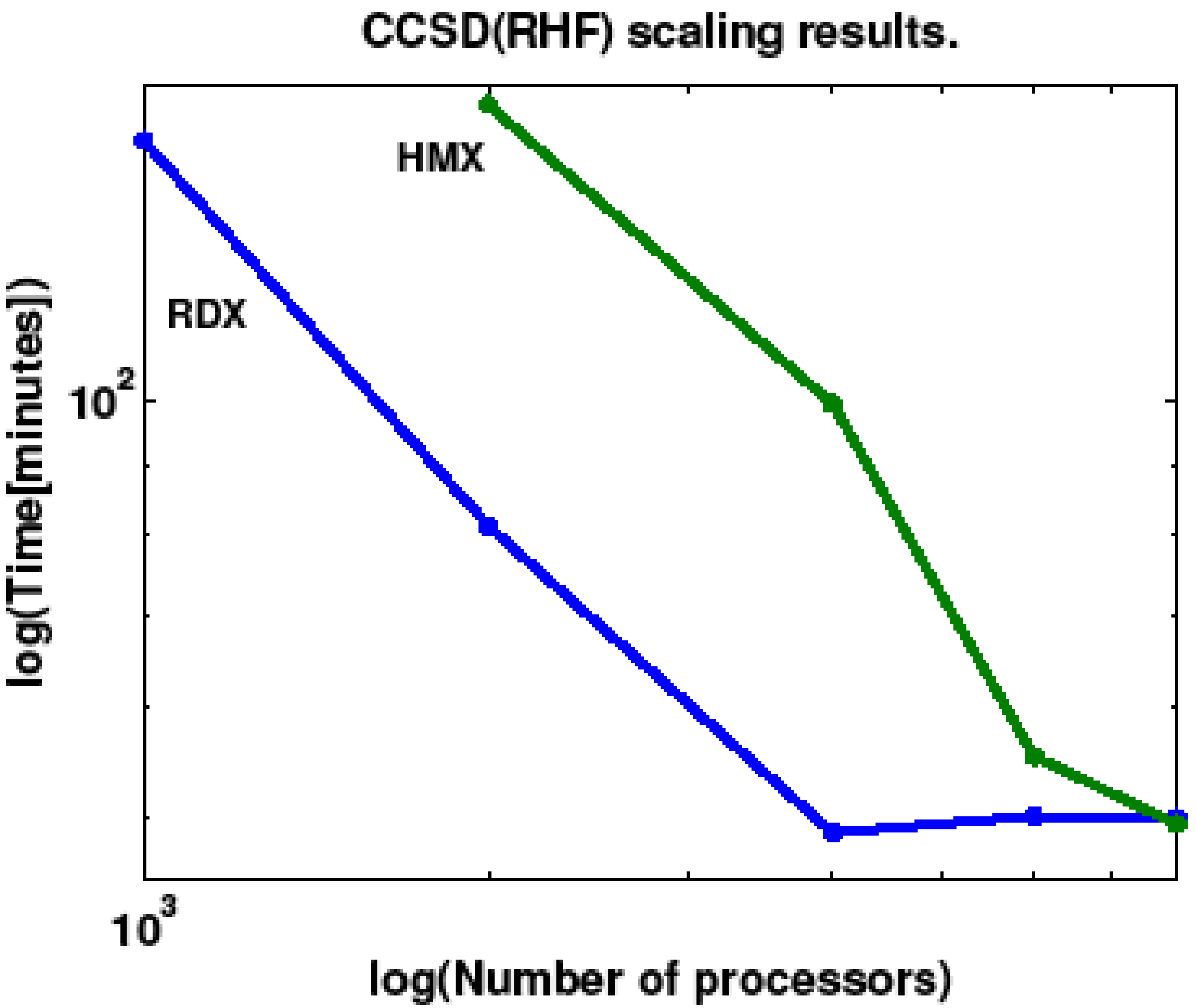
MP2 gradient(RHF) scaling results.

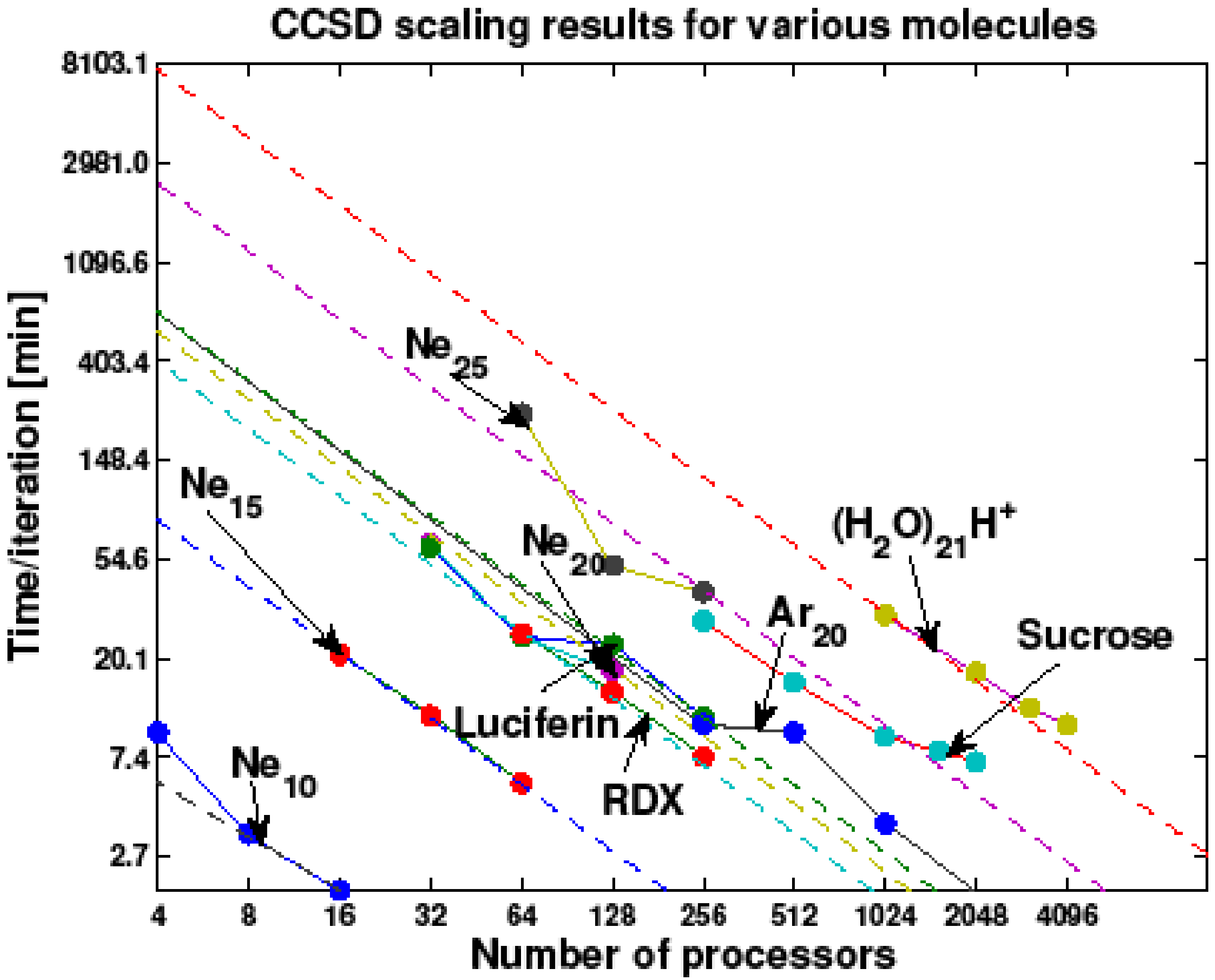


Computational details of CCSD computation

Method	CCSD	CCSD
Molecule	RDX	HMX
Number of bf’s	1005	924
Number of electrons	114	152
Number of atoms	21	28


CCSD(RHF) scaling results.
HMX
RDX
Time[minutes]
40
60
80
100
120
140
160
1000
2000
3000
4000
5000
6000
7000
8000
Number of processors

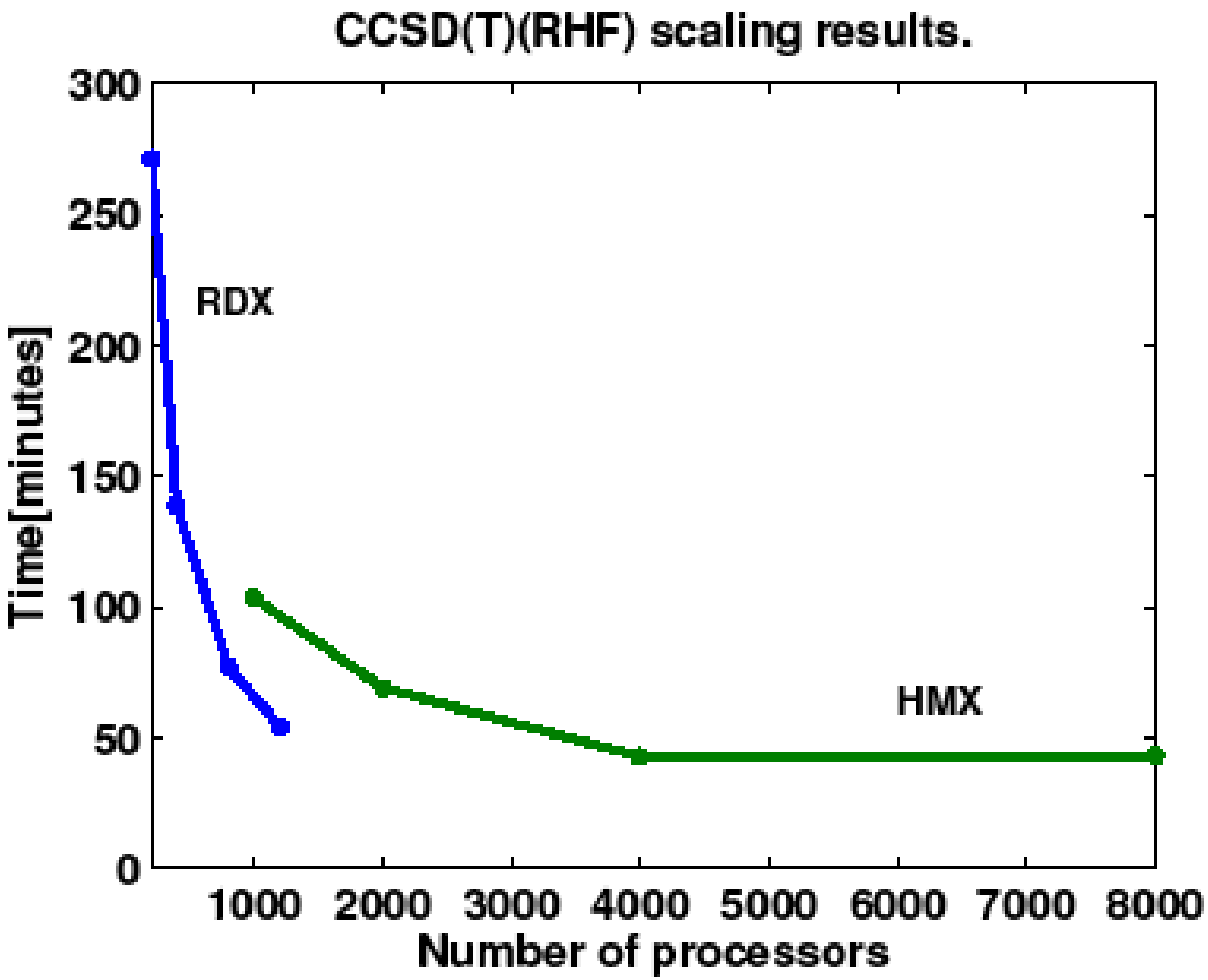

CCSD(RHF) scaling results.
HMX
RDX
10^2
10^3
log(Time[minutes])
log(Number of processors)



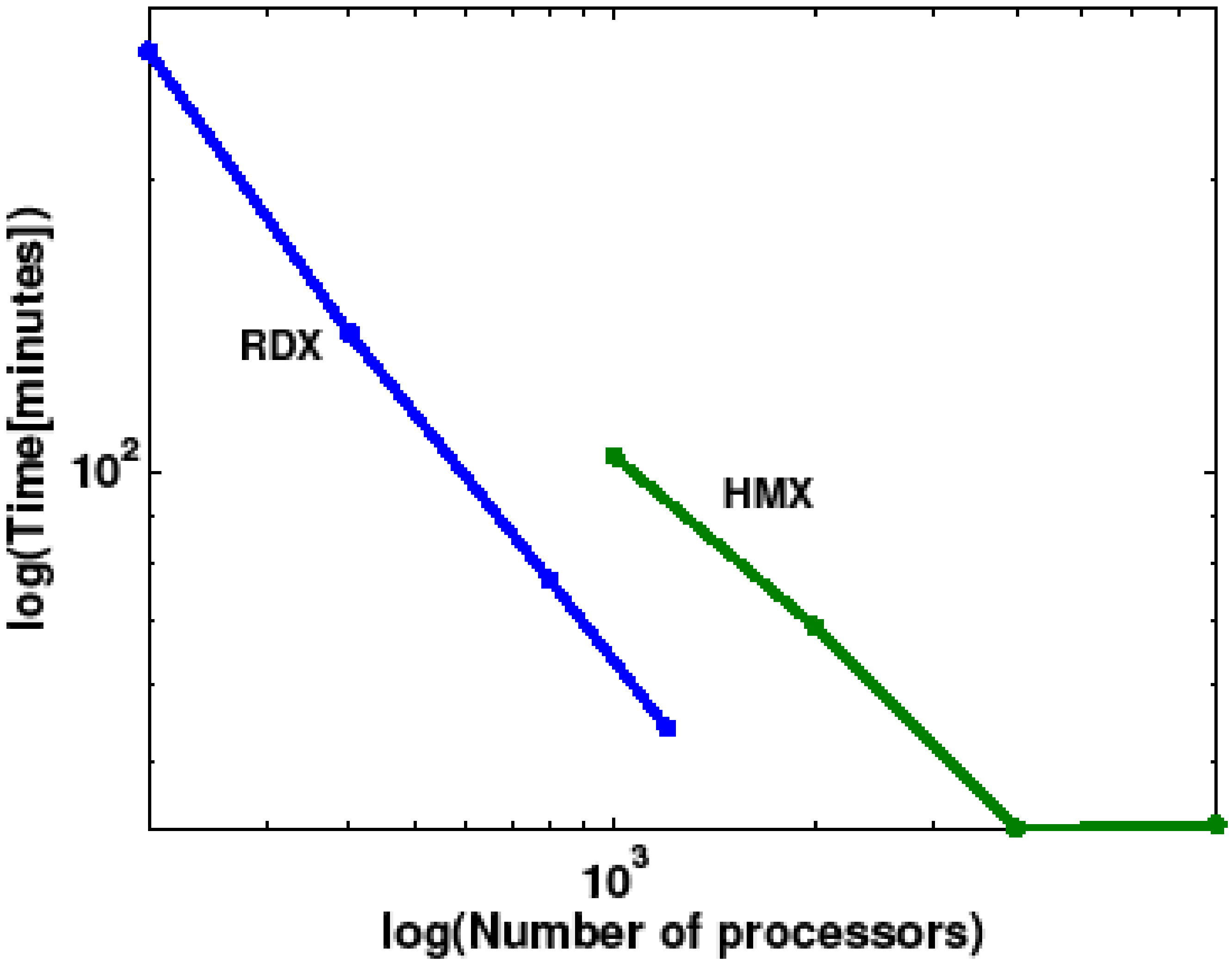
CCSD scaling results for various molecules
Time/iteration [min]
8103.1
2981.0
1096.6
403.4
148.4
54.6
20.1
7.4
2.7
4
8
16
32
64
128
256
512
1024
2048
4096
Number of processors
Ne_{25}
Ne_{15}
Ne_{20}
$(H_2O)_{21}H^+$
Ar_{20}
Sucrose
Luciferin
RDX
Ne_{10}

Computational details of CCSD(T) computation

Method	CCSD(T)	CCSD(T)
Molecule	RDX	HMX
Number of bf’s	372	496
Number of electrons	114	152
Number of atoms	21	28

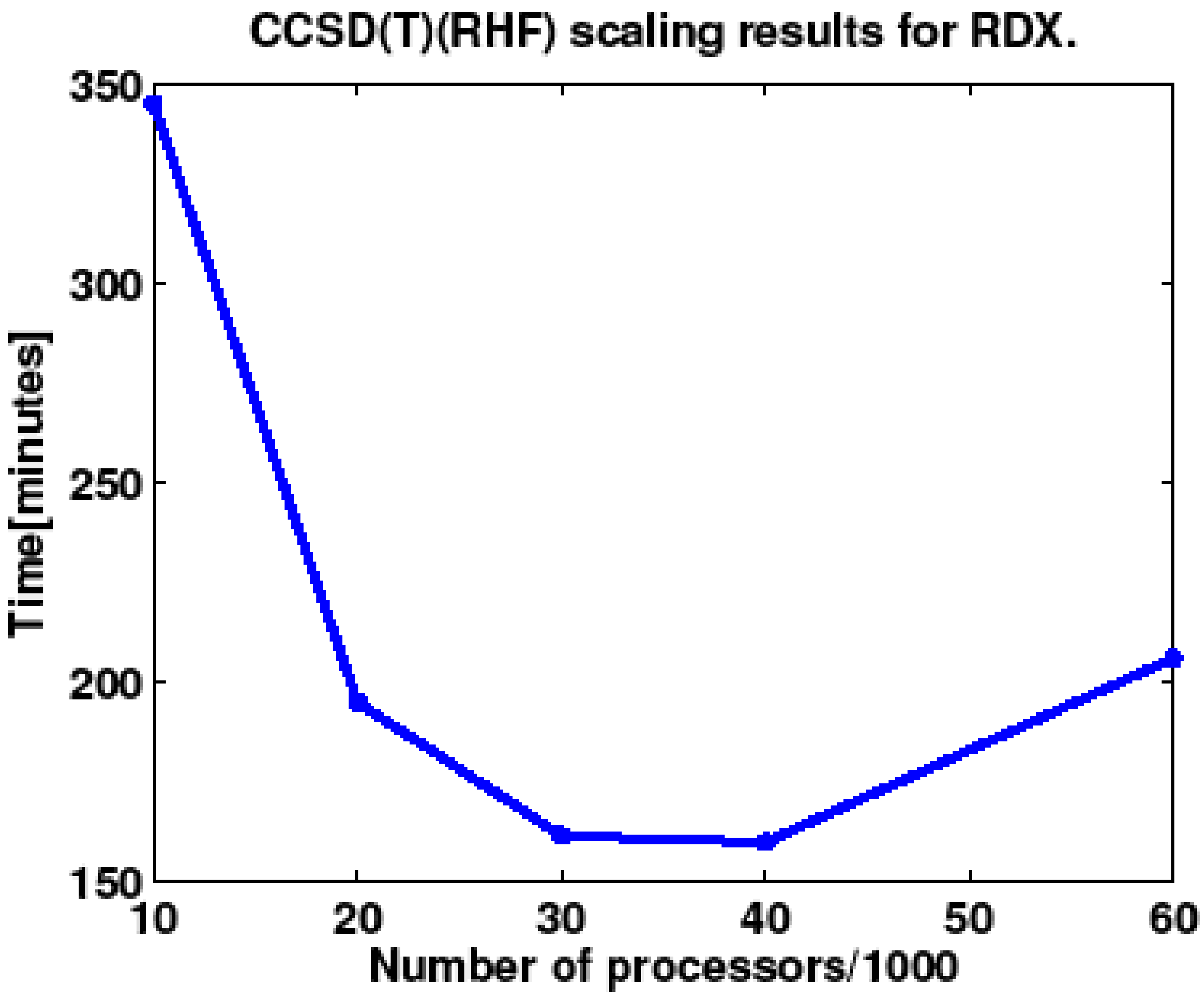

CCSD(T)(RHF) scaling results.
Time[minutes]
Number of processors
RDX
HMX
0
50
100
150
200
250
300
1000
2000
3000
4000
5000
6000
7000
8000

CCSD(T)(RHF) scaling results.

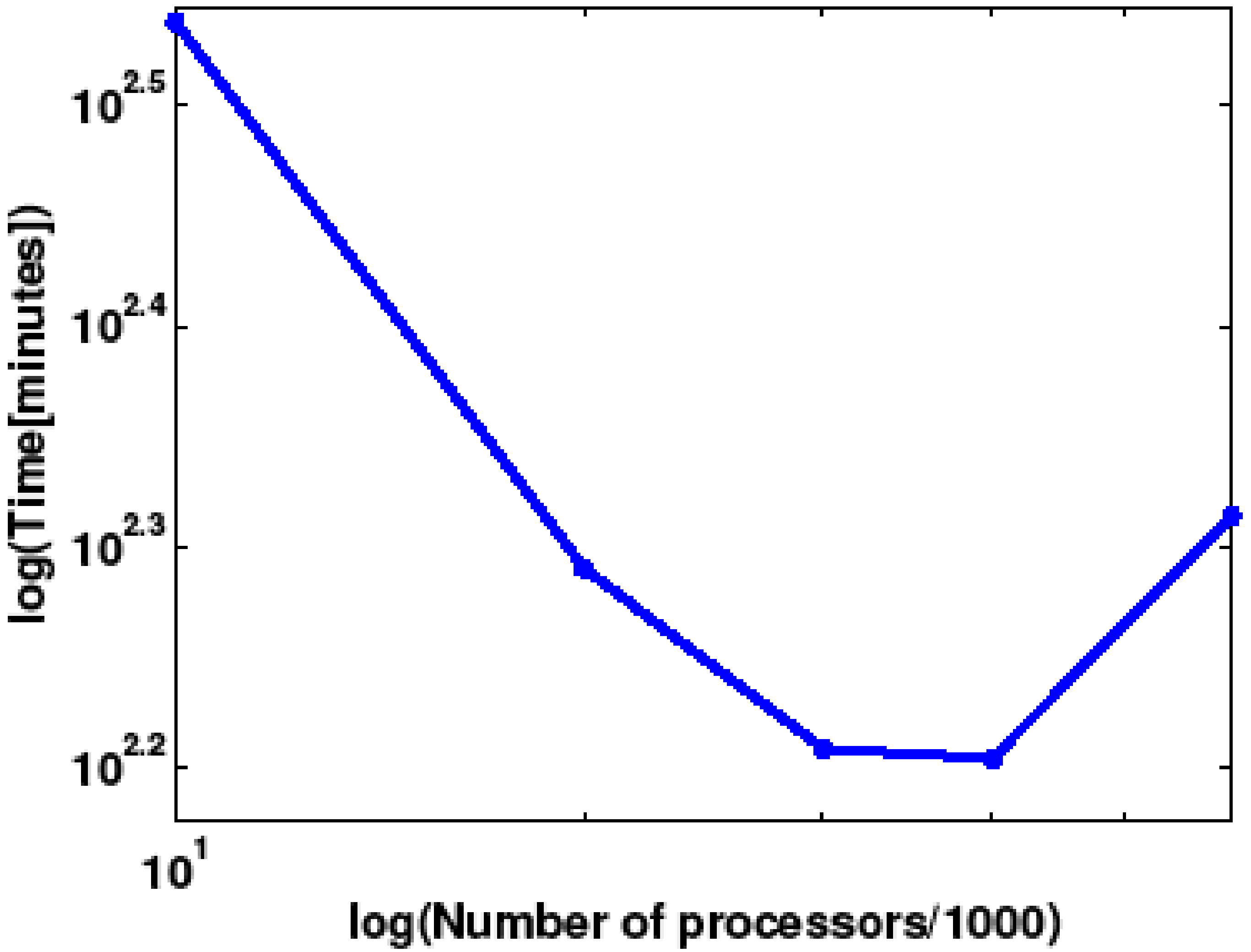


Computational Details

Method	CCSD(T)
Molecule	RDX
Number of basis functions	1005
Number of electrons	114
Number of atoms	21


CCSD(T)(RHF) scaling results for RDX.
Time[minutes]
Number of processors/1000
350
300
250
200
150
10
20
30
40
50
60

CCSD(T)(RHF) scaling results for RDX.



CONCLUSIONS

- The AcesIII framework has allowed many quantum chemistry codes to be written in parallel with good-excellent scaling.
- The separation of efficiency/algorithmic aspects leads to a more productive programming environment.
- The generality of SIP/SIAL allows for other disciplines to use AcesIII.

Improvements

- Subindex capability: segments can be further subdivided in to subindeces.
- Data mining: different sections of code can be assigned a different set of processors to run on.
- High dimension arrays(<=10) $\longrightarrow$ compound indices **NOT TRIVIAL**