

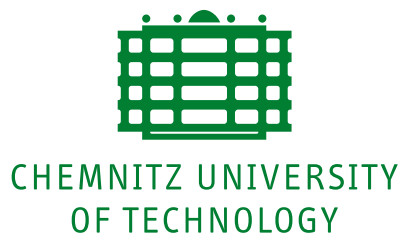
Computer Architecture Technical Report

TUC / RA-TR-2005-02

Date: 16th Jul 2005
(Last Build: 22nd July 2005)

A short Performance Analysis of Abinit on a Cluster System

Torsten Hoeffler, Wolfgang Rehm
{htor,rehm}@informatik.tu-chemnitz.de



Chemnitz University of Technology
Department of Computer Science
Computer Architecture
Prof. Dr. W. Rehm

A short Performance Analysis of Abinit on a Cluster System

Torsten Hoefler, Wolfgang Rehm

{h^tor,rehm}@informatik.tu-chemnitz.de

Keywords

Ab Initio, Quantum Mechanics, Benchmarks, Scaling, Cluster

1 Introduction

Abinit [1] uses the Density Functional Theory (DFT) to perform atomic scale simulation of nano-structures. The code offers three different methods for MPI parallelization:

1. parallel calculation of different k-points (points in the inverse space with the same energy level)
2. parallel calculation of different bands
3. parallel FFT (fast Fourier Transformation)

The parallelization method is either chosen by defining preprocessor flags (such as `-DMPI` or `-DMPI_FFT`) or by input parameters (such as `wfoptalg` or `nbdblock`, see the Documentation of Abinit for further details). All different parallelization methods are investigated with regards to their running time and scalability on cluster systems.

2 Benchmark Environment

Our local cluster, consisting of 8 Dual Xeon 2.4Ghz is used for benchmarking. Each node has 2GB main memory and a I/O disk system with an approximate bandwidth of 50MB/s. The nodes are interconnected with Gigabit Ethernet. MPICH2 1.0.2p1 [2] was used as MPI library. The used version of Abinit was 4.5.2 compiled with the Intel Fortran Compiler 8.1.

2.1 Compiling Abinit

The Software was compiled with the Intel 8.1 Compiler (Build 20050520Z). All relevant entries of the `makefile_macros` are shown in the following:

```
FC=ifort
COMMON_FFLAGS=-FR -w -tpp7 -axW -ip -cpp
FFLAGS=$(COMMON_FFLAGS) -O3
FFLAGS_Src_2psp=$(COMMON_FFLAGS) -O0
FFLAGS_Src_3iovars=$(COMMON_FFLAGS) -O0
FFLAGS_Src_9drive=$(COMMON_FFLAGS) -O0
FFLAGS_LIBS=-O3 -w
FLINK=-static
```

The `-O3` optimization had to be disabled for several directories, due to compiler bugs which led to endless compiling.

2.2 Benchmark Input File

All sequential, FFT and k-point parallelized benchmarks have been executed with a mainly identical input file which defines a unit-cell with 14 atoms. The number of k-points was changed with the `ngkpt` parameter as shown in the following table.

#kpt	ngkpt
2	2 2 2
4	4 2 2
8	4 4 2
16	4 4 4

To use parallelization over bands, it was necessary to change the default wavefunction optimisation algorithm (`wfoptalg` was changed to 1) and the number of bands in a block (`nbdblock` was changed as described in section 3.2 ¹) parameters. The SCF parameter `toldfe` was changed to $1.0d - 11$ to force a sufficient convergence, this resulted in 21 SCF Cycles (the band parallelization usually used more iterations due to worse convergence).

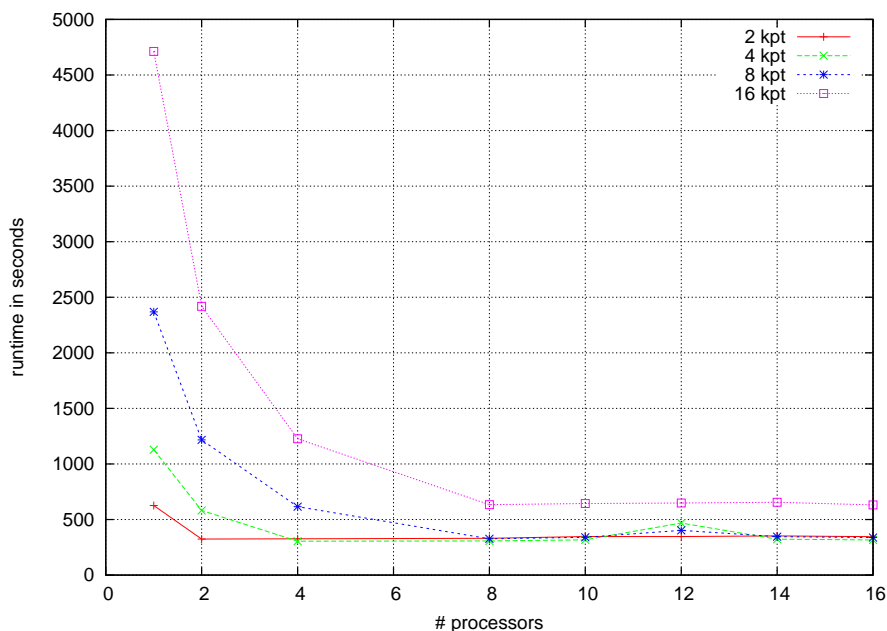
3 Benchmark Results

The benchmark results are presented in the following sections for each parallelisation method up to 16 processors.

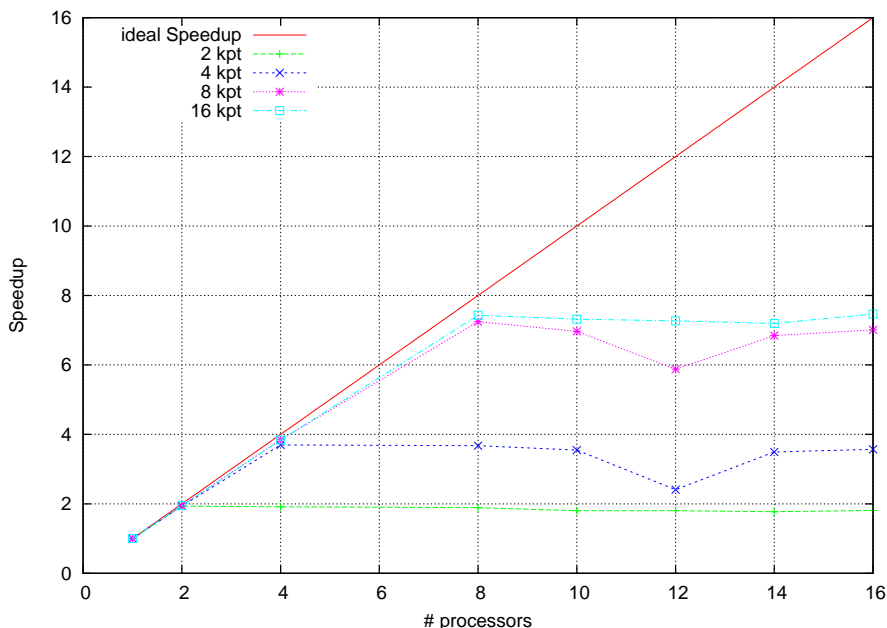
3.1 Plain k-point Parallelization

The following table shows the overall execution time of the parallel version with a different number of k-points on a varying number of CPUs in seconds. It shows two values for each k-pt to nodenumber combination, the first one give the number of seconds needed to perform three SCF cycles and the second one the number of seconds to full convergence.

#CPUs \ #kpt	2	4	8	16
1	137/626	268/1128	537/2369	1065/4712
2	72/324	140/582	277/1216	550/2418
4	72/327	74/305	143/616	287/1227
8	72/331	74/307	78/327	153/634
10	76/347	77/318	81/340	155/644
12	76/347	80/468	78/403	155/648
14	77/353	78/323	84/346	156/655
16	75/346	76/316	81/338	154/631



¹see <http://www.abinit.org/wws/arc/forum/2003-02/msg00004.html> for details



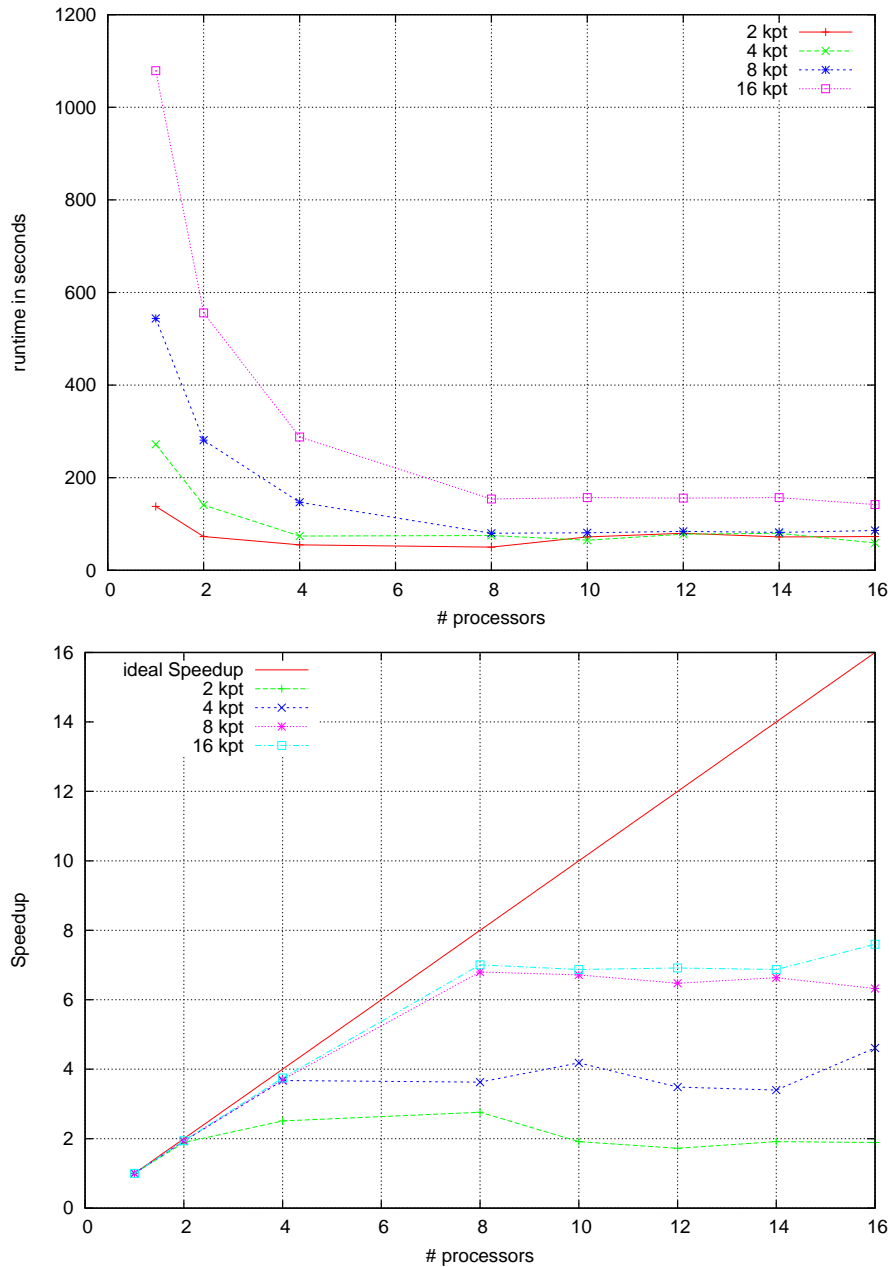
The runtime graph and the speedup (shown for full convergence) shows that the k-pt parallelization scales very fine up to the actual number of k-points. This parallelization is only efficient for a sufficient number of k-points (should be a multiple of the number of nodes). We also discover an upper border to the scalability with multiple k-points at 8. Even for 16 k-points and 16 processors, the speedup did not grow to more than eight. But in general can be said that the parallelization over k-points is very efficient under the above mentioned limits ($n_{CPU} = m \cdot n_{kpt}$, $m \in \mathbb{N}$). This fact limits the useability of the k-point parallelization in Abinit to small molecular systems² with many k-points.

3.2 Parallelization over Bands

This method parallelizes over k-points and bands. It is able to utilize more processors due to additional parallelization, but the speedup may not be ideal because the band-parallelized method converges not as good as the plain k-point parallelization. Additional SCF steps are needed to achieve the same results. The following table shows the overall running time depending on the number of k-points and the number of nodes. The parameter `nblock` was chosen to fit the criterion $nblock = \frac{n_{CPU} \cdot x}{n_{kpt}}$, $x \in \mathbb{N}$ this leads in most cases to efficient work distribution. However, there are several cases, e.g. for 4 k-points on 10 processors, where no ideal fit could be found - the measurements were taken with non idel distributions in these particular cases.

#CPU\#kpt	2	4	8	16
1	138/633	272/1145	544/2396	1079/4773
2	73/330	141/589	281/1233	556/2437
4	55/260	74/305	147/626	288/1241
8	50/238	75/307	80/331	154/640
10	72/414	65/360	81/329	157/656
12	80/467	78/402	84/344	156/656
14	72/379	80/439	82/319	157/662
16	73/493	59/327	86/385	142/634

²usually the number of needed k-points decreases with an increasing number of atoms, e.g. systems with more than 20 atoms need only 1 k-point



The band parallelism is not able to enhance the speedup in this configuration. This may be due to the small number of bands `nband=56` or to additional SCF iterations. This parallelization is not able to speedup the calculation of our specified system on our cluster.

3.3 Parallel FFT

The parallel FFT implementation is quite new and not documented yet. Abinit v4.5.2 has also a bug which prevents the code from being run (it calls `MPI_Comm_Free(MPI_COMM_SELF)`). After removing this bug, the code runs stable for 1 node, but for more nodes, one of the following errors occurs:

```

irrzg : BUG -
  ifft,irrzon(ifft,1,imagn),nfftot,imagn= 1 0 155520 1
=>irrzon goes outside acceptable bounds.
Action : contact ABINIT group.

```

or:

```

getng : BUG -
The third dimension of the FFT grid, ngfft(3), should be
a multiple of the number of processors for the FFT,

```

```
nproc_fft. However, ngfft(3)= 30 and nproc_fft= 4
Action : contact ABINIT group.
```

The code seems to be still buggy, and it was not possible to run any benchmark with more than one node.

4 Short Parallel Analysis

We used the MPE environment and Jumpshot [3] to make a short analysis of the communication behavior of abinit in the different working scenarios.

4.1 Plain k-point Parallelization

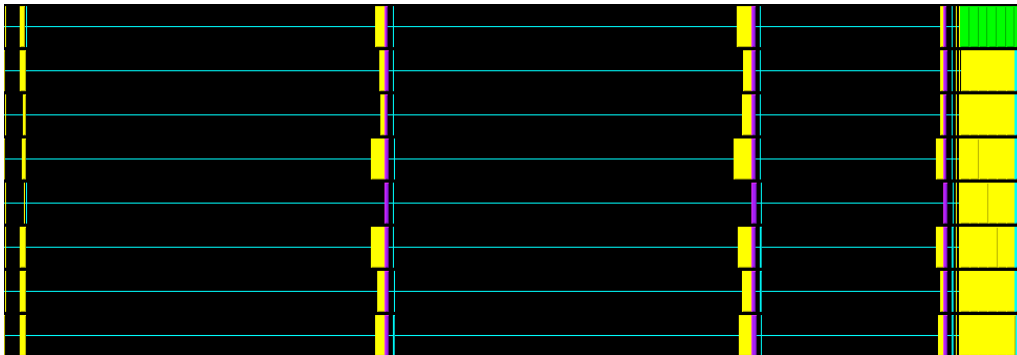
This parallelization method is investigated in two scenarios, the nearly ideal speedup with 8 processors calculating 8 k-points and the not ideal speedup with 16 processors calculating 16 k-points. The MPI communication scheme of 8 processors calculating 8 k-points is shown in the following picture. The processors are shown on the ordinate (rank 0-7), and the communication operations are shown for each of them. Each MPI operation corresponds with a different color, they are namely:

MPIBarrier yellow

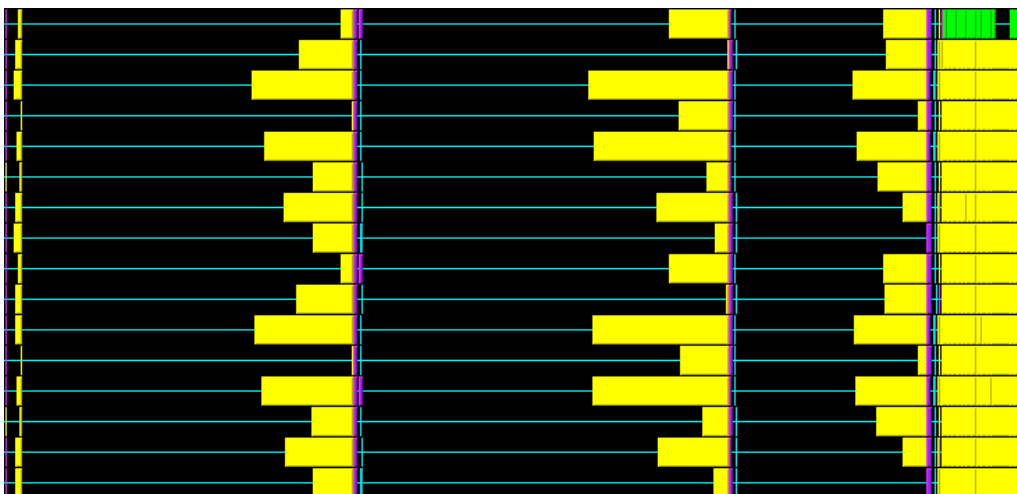
MPIRecv green

MPIAllreduce lavender

MPIBcast light blue

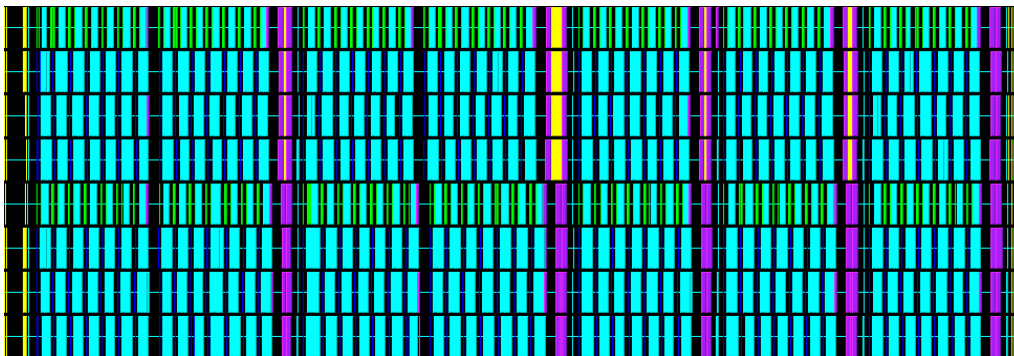


The scheme for 16 processors calculating 16 k-points is identical, but the overhead resulting from barrier synchronization is much higher and decreases the performance.

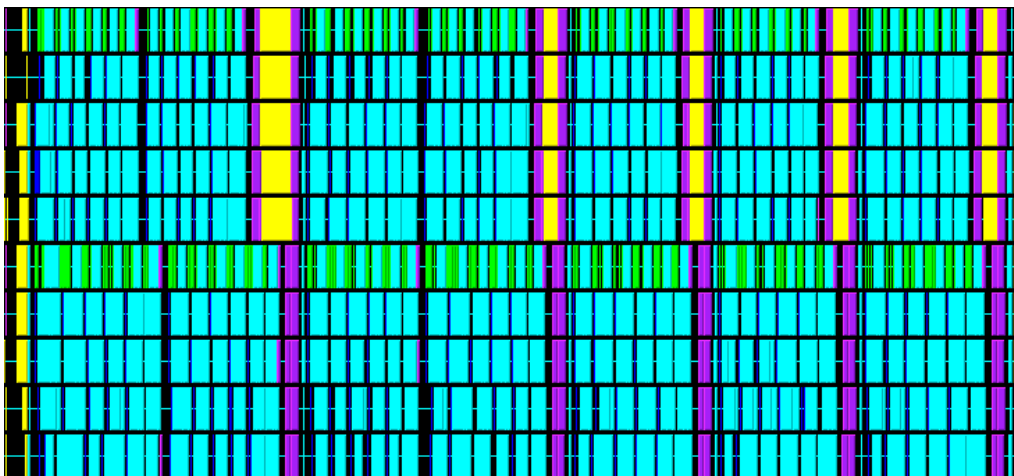


4.2 Parallelization over Bands

The parallelization over the bands seems quite unsatisfying in this particular use case. The communication diagram for 8 processors calculation 2 k-points is shown in the following picture.



The communication intensity increased, and the MPI_Bcast calls dominate. The MPI overhead (mainly due to MPI_Bcast) reaches up to 50% in this case. The overall running time was even increased, when more processors were added to process the job (negative speedup). To analyze this behavior, a communication graph of 10 processors calculating the 2 k-points is shown in the following.



The additional nodes cause some imbalance which increases the synchronization overhead. This causes the negative speedup in this particular case.

5 Summary and Conclusion

This short analysis shows, that the Abinit program package cannot be used to calculate unit cells with many atoms on large scale cluster systems. The speedup which can be achieved with k-point parallelization is excellent up to the number of k-points in the system or when the nodes suffer from synchronization problems (beginning with 8 nodes). The band parallelization offers an additional method to scale to a bigger number of nodes but increases the communication costs and does not improve the speedup on our system. The FFT parallelization is not fully implemented yet and stops with internal errors. This study shows that Abinit does not scale to more than 8 processors with our specific use case (14 atoms, 56 bands, up to 16 k-points).

References

- [1] X. GONZE, J.-M. BEUKEN, R. CARACAS, F. DETRAUX, M. FUCHS, G.-M. RIGNANESE, L. SINDIC, M. VERSTRAETE, G. ZERAH, F. JOLLET, M. TORRENT, A. ROY, M. MIKAMI, PH. GHOSEZ, J.-Y. RATY AND D.C. ALLAN: *First-principles computation of material properties: the ABINIT software project*, *Computational Materials Science* 25, 478-492 (2002), see www.abinit.org
- [2] WILLIAM GROPP: *A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard*, *Intl. Journal of Parallel Computing*, Vol. 22, Nr. 6, pp 789-828, 1996
- [3] OMER ZAKI, EWING LUSK, WILLIAM GROPP AND DEBORAH SWIDER: *Toward Scalable Performance Visualization with Jumpshot*, *In The International Journal of High Performance Computing Applications*, Vol. 13, Nr. 3, pp 277-288, 1999