ABINIT ABINIT ABINIT ABINIT

Y. Pouillon

Université Catholique de Louvain - Louvain-la-Neuve, Belgium

ABINIT Summer School
Santa Barbara, CA, USA
2005/08/26

Outline

1. Introduction to parallelism
   - Parallel environments
   - A brief overview of MPI
2. Parallelism inside ABINIT
   - Currently available
   - Ongoing efforts

Outline

Types of parallelism

Parallel computing

1. Work done in separate processes
2. Data exchange for communication

- SIMD: Single-Instruction, Multiple-Data (Data-Parallel)
  - homogeneous environments
  - examples: SMP machines, HPF, OpenMP
- MIMD: Multiple-Instruction, Multiple-Data
  - heterogeneous environments
  - examples: clusters, MPI
- Either distributed or shared memory
- Data transfer one-sided or cooperative

Types of parallelism

Parallel computing

1. Work done in separate processes
2. Data exchange for communication

- SIMD: Single-Instruction, Multiple-Data (Data-Parallel)
  - homogeneous environments
  - examples: SMP machines, HPF, OpenMP
- MIMD: Multiple-Instruction, Multiple-Data
  - heterogeneous environments
  - examples: clusters, MPI
- Either distributed or shared memory
- Data transfer one-sided or cooperative

Popular environments

- OpenMP:
  - shared-memory parallel platforms
  - simple, flexible, portable, scalable
  - supported by many hardware & software vendors
  - ABINIT support withdrawn from 4.2 version
- MPI: Message-Passing Interface
  - general-purpose parallel API
  - flexible, portable, scalable
  - standard designed by the MPI Forum (`http://www.mpi-forum.org/`)
  - standard versions: MPI-1 (1994), MPI-2 (1997)
  - most parallel parts of ABINIT

Exercise: a simple algorithm

- Initialization: number of neighbours you have
- Compute average of your neighbours' values
- Subtract yours
- Repeat until done

Exercise: a simple algorithm

- Initialization: number of neighbours you have
- Compute average of your neighbours' values
- Subtract yours
- Repeat until done

Questions

1. How do you get values from your neighbours?
2. Which step or iteration do they correspond to? Do you know? Do you care?
3. How do you decide when you are done?

Outline

MPI: Message-Passing Interface

- Normalized functions for inter-process communications
- Heterogeneous environments
  $\Longrightarrow$ portability of source code
- Easy-to-use, comprehensive and powerful
- Each process works on local data
- Data shared by sending/receiving “messages”
- C and Fortran supported
- Developer’s tasks:
  - balance the load between processes
  - minimize communication
  - “superpose” work and communications

Basic concepts

- All processors execute the same code
- Communications:
  - peer-to-peer: involving a pair of processes
  - collective: involving a group of processes
- All MPI functions start with `MPI_`
- All MPI functions return an error code (= `MPI_SUCCESS` if OK)
- Definition of portable types:
  - `MPI_INTEGER`
  - `MPI_REAL`
  - ...

Parallelizing a code: a silly example

Sequential

- hello_seq.f90:

```
program hello_seq

 write (*,*) "Hello, world!"

end program hello_seq
```

- `f90 -o hello_seq hello_seq.f90`
- `./hello_seq`

Parallel

- hello_par.f90:

```
program hello_par

#include "mpif.h"

 integer :: err

 call MPI_INIT(err)
 write (*,*) "Hello, world!"
 call MPI_FINALIZE(err)

end program hello_par
```

- ```
  f90 -I/usr/lib/lam/include \
      -L/usr/lib/lam/lib -lmpi \
      -o hello_par hello_par.f90
  ```
- `mpirun -np 4 hello_par`
  ```

Parallelizing a code: a silly example

Sequential

- hello_seq.f90:

```
program hello_seq

 write (*,*) "Hello, world!"

end program hello_seq
```

- `f90 -o hello_seq hello_seq.f90`
- `./hello_seq`

Parallel

- hello_par.f90:

```
program hello_par

#include "mpif.h"

 integer :: err

 call MPI_INIT(err)
 write (*,*) "Hello, world!"
 call MPI_FINALIZE(err)

end program hello_par
```

- ```
  f90 -I/usr/lib/lam/include \
      -L/usr/lib/lam/lib -lmpi \
      -o hello_par hello_par.f90
  ```
- `mpirun -np 4 hello_par`
  ```

All-in-one

Using only one source file

- hello.F90:

```
program hello

#if defined MPI
#include "mpif.h"

 integer :: err
#endif

#if defined MPI
 call MPI_INIT(err)
#endif
 write (*,*) "Hello, world!"
#if defined MPI
 call MPI_FINALIZE(err)
#endif

end program hello
```

- `f90 -o hello_seq hello.F90`
-
```
f90 -DMPI \
    -I/usr/lib/lam/include \
    -L/usr/lib/lam/lib -lmpi \
    -o hello_par hello_par.f90
```
- `./hello_seq`
- `mpirun -np 4 hello_par`

Outline

1. Introduction to parallelism
   - Parallel environments
   - A brief overview of MPI
2. Parallelism inside ABINIT
   - Currently available
   - Ongoing efforts

K-point & spin parallelism

- Very efficient: few communications needed
- Optimal on periodic systems requiring a lot of resources
- Set-up automatically by *abinip*
- Ideal # of processors: *nkpt* $\times$ *nspden*
- Can be mixed with band parallelism
- Few limitations

Band parallelism

- Systems with a lot of bands & few k-points
  $\longrightarrow$ typically: molecules in big boxes (1 k-point)
- Controlled by *wfoptalg* & *nbdblock*
- Not tested beyond *nbdblock* = 4
  (not expected to be efficient though)
- Recommended values:
  - *wfoptalg* = 1
  - *nbdblock* = 2–4
- Typical # of processors: 4–8

Outline

1. Introduction to parallelism
   - Parallel environments
   - A brief overview of MPI

2. Parallelism inside ABINIT
   - Currently available
   - Ongoing efforts

Parallel FFT

- Still very experimental in ABINIT
- Activated by `-DMPIFFT` at compile-time
- Controlled by *fft_opt_lob* input variable
- Other way: take benefit from existing libraries, e.g. FFTW

Parallel I/O

- POSIX: filesystem portability and optimization
  $\longrightarrow$ only for sequential programs
- MPI-I/0
  - partitioning of file data among processes
  - data transfer between process memories and files
  - asynchronous I/O, strided access
  - control over physical file layout on disks
  - original model based on derived datatypes
    (instead of I/O access modes)
- Implementation in ABINIT still very incomplete
- Activated by `-DMPIO` at compile-time

Summary

- Many parallel environments available
- Parallelize a code is a delicate operation
- MPI: one of the most popular, highly portable
  $\longrightarrow$ choice for ABINIT
- ABINIT is already parallelized wrt:
  - k-points / spin-polarization
  - bands (less efficient)
  - specific tasks (e.g. TDDFT)

  Ongoing efforts include:
  - parallel FFT
  - parallel I/O (MPIO)

References

- Section on parallel computing from Wikipedia
  http://en.wikipedia.org/wiki/Parallel_computing
- MPI Forum website
  http://www.mpi-forum.org/
- Open MPI project website
  http://www.open-mpi.org/
- MPI tutorial from Argonne National Laboratory
  http://www-unix.mcs.anl.gov/mpi/tutorial/
- MPI tutorials from the LAM-MPI project
  http://www.lam-mpi.org/tutorials/
- MPI par l'exemple
  http://bigbrother.pcpm.ucl.ac.be/mpi/mpi.html

Acknowledgments

- S. Dubois, M. Verstraete, X. Gonze
- CISM (center for high-performance computing in Louvain-la-Neuve)
- ABINIT community

Thank you for your time!

Acknowledgments

- S. Dubois, M. Verstraete, X. Gonze
- CISM (center for high-performance computing in Louvain-la-Neuve)
- ABINIT community

Thank you for your time!