Introduction to Parallel Computing

Presented by
Carol Gauthier

Centre de Calcul Scientifique (CCS)
Université de Sherbrooke

Réseau québécois de calcul de haute performance (RQCHP)
Introduction to Parallel Computing

Summary of the presentation

1) Overall Performance Strategy
2) Goals of Parallel Computing
3) Parallel Computer Architectures
4) Parallelization Strategy
5) Shared Memory Parallel Programming
6) Distributed Memory Parallel Programming
Introduction to Parallel Computing

Overall Performance Strategy

Scientific Application Development Steps

<table>
<thead>
<tr>
<th>Step</th>
<th>Timeframe</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{sa}$</td>
<td>1-12 months</td>
</tr>
<tr>
<td>$T_{spec}$</td>
<td>1-6 months</td>
</tr>
<tr>
<td>$T_{code}$</td>
<td>1-12 months</td>
</tr>
<tr>
<td>$T_{opt}$</td>
<td>1-6 weeks</td>
</tr>
<tr>
<td>$T_{par}$</td>
<td>1-6 months</td>
</tr>
<tr>
<td>$T_{prep}$</td>
<td>1 msec-60 min / run</td>
</tr>
<tr>
<td>$T_{run}$</td>
<td>1 min – x months / run</td>
</tr>
<tr>
<td>$T_{mod}$</td>
<td>1-12 months</td>
</tr>
</tbody>
</table>

Scientific Analysis
Theory, Maths, Algorithms, Data Structures, etc.

Writing Specifications

Coding

Serial Optimization

Parallelization (if necessary)

Prepare cases

Run cases

Code Modifications

Cases loop
Modification loop

Carol Gauthier, Centre de Calcul Scientifique (CCS), Université de Sherbrooke
Even after an excellent parallelisation or optimisation...

... this code will never run more than 3 times faster !!!
Overview of Development and Production Time

**Development**

- **Tsa**
- **Tsdesc**
- **Tcoding**
- **Topt**
- **Tpar**

**Production**

- **Tprep**
- **Trun**
- **Tmodif**
Larger and Longer Cases

Solve larger cases
*Parallelisation will allow to alleviate the memory limitations of a single processor (or a single node or workstation).*

Solve each problem case faster
*Parallelisation will allow to solve each case of your problem within a reasonable amount of time.*

“If each case of your problem is solved within the memory of a node, and within a reasonable amount of time using a single processor, there is no need to parallelize!”
Introduction to Parallel Computing

Parallel Computer Architectures

1) Shared Memory Architectures

2) Distributed Memory Architectures: Clusters

3) Hybrid Architectures
Shared Memory Architectures

Single memory image

Your program in memory

Another program

Memory Switch Fabric

CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU CPU
Distributed Memory Architectures: Clusters

Your program in memory task 0

System

CPU

Network Card

Memory Node 1

Your program in memory task 1

System

CPU

Network Card

Memory Node 2

Your program in memory task 2

System

CPU

Network Card

Memory Node 3

Your program in memory task 3

System

CPU

Network Card

Memory Node 4

Network Switches
Introduction to Parallel Computing
Parallel Computer Architectures

Hybrid Architectures
Introduction to Parallel Computing

Parallelization Strategy

1. Initial Optimized Serial Code
2. Setup a medium size reference case (RefCase)
3. Run the RefCase - Measure overall runtime - Save results (RefResult)
4. Determine the Architecture and the Parallel Model
5. Implements the Parallel Model
6. Improve and Optimize Parallelization Communication - Load Balancing - Sychronisation
7. Measure Performance and Scaling
8. Parallized Code

Optimization loop
Introduction to Parallel Computing
Shared Memory Parallel Programming

1) Shared Memory Programming Interfaces
2) OpenMP: Accessing Multiple Threads
3) OpenMP: Directives
4) OpenMP: Subroutines and Functions
5) OpenMP: Environment Variables
6) OpenMP: Examples
7) OpenMP: False Sharing!
OpenMP

The standardized API for multithreaded parallel applications

A set of directives and functions
C$OMP PARALLEL, #pragma omp parallel ,etc...

Available for Fortran / C / C++

Simple to use compare to other API (POSIX Threads)
A thread is a sequence of operations that is executed by the system. There are no priority or dependance among threads, so the order in which they will be executed is not known! Two threads belonging to a single instance of a program, could then be executed at the same time if several CPU are available to the system.
OpenMP: Accessing Multiple Threads
OpenMP: Directives

**In Fortran (fixed source form)**

```
!$OMP directive_name [clause [[,] clause] ....]
```

or

```
C$OMP directive_name [clause [[,] clause] ....]
```

or

```
*$OMP directive_name [clause [[,] clause] ....]
```

**In Fortran (free source form)**

```
!$OMP directive_name [clause [[,] clause] ....]
```

**In C/C++**

```
#pragma omp directive-name [clause[ clause] ....]
```
OpenMP: Directives

**Fortran**

```fortran
!$OMP PARALLEL
   parallel code ...
!$OMP END PARALLEL

!$OMP DO
!$OMP END DO

!$OMP SECTIONS
   !$OMP SECTION
   !$OMP SECTION
!$OMP END SECTIONS

!$OMP SINGLE
!$OMP END SINGLE

!$OMP MASTER
!$OMP END MASTER

.... and more...
```

**C/C++**

```c
#pragma omp parallel
  structured-block

#pragma omp for
  for-loop

#pragma omp sections
  {
    [#pragma omp section]
    structured-block
    [#pragma omp section]
  structured-block

#pragma omp single
  structured-block

#pragma omp master
  structured-block

.... and more...
```
## Introduction to Parallel Computing

### Shared Memory Parallel Programming

## OpenMP: Useful Subroutines and Functions

<table>
<thead>
<tr>
<th><strong>Fortran</strong></th>
<th><strong>C/C++</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functions</strong></td>
<td><strong>Functions</strong></td>
</tr>
<tr>
<td>omp_get_num_threads()</td>
<td>omp_get_num_threads()</td>
</tr>
<tr>
<td>omp_get_max_threads()</td>
<td>omp_get_max_threads()</td>
</tr>
<tr>
<td>omp_get_thread_num()</td>
<td>omp_get_thread_num()</td>
</tr>
<tr>
<td>omp_get_num_procs()</td>
<td>omp_get_num_procs()</td>
</tr>
<tr>
<td>omp_in_parallel()</td>
<td>omp_in_parallel()</td>
</tr>
<tr>
<td><strong>Subroutines</strong></td>
<td><strong>Subroutines</strong></td>
</tr>
<tr>
<td>call omp_set_num_threads(int)</td>
<td></td>
</tr>
</tbody>
</table>
OpenMP: Environment Variables

**OMP_NUM_THREADS**
Number of threads during the run

**OMP_SCHEDULE**
Type of scheduling in DO directives (static, dynamic, guided)

**OMP_DYNAMIC**
Enable/disable dynamic adjustments of the number of threads

**OMP_NESTED**
Enable/disable nested parallelism
OpenMP: An example Using !$OMP PARALLEL and !$OMP DO

... lines of codes

 !$OMP PARALLEL
 ... lines of codes

 !$OMP DO
 do k=1,10000
  do j=1,10000
   do i=1,5000
    a(i,j,k)=c(i)*k - b(j)
   end do
  end do
 end do
 !$OMP END DO

... lines of codes

 !$OMP PARALLEL
 ... lines of codes

Carol Gauthier, Centre de Calcul Scientifique (CCS), Université de Sherbrooke
Introduction to Parallel Computing
Shared Memory Parallel Programming

OpenMP: An example Using !$OMP PARALLEL DO

... lines of codes

 !$OMP PARALLEL DO
 do k=1,10000
   do j=1,10000
     do i=1,5000
       a(i,j,k)=c(i)*k - b(j)
     end do
   end do
 end do
 !$OMP PARALLEL DO

... lines of codes

One master thread (serial code)
Entering parallel region

} Each thread is working with different k values
Exciting parallel region

One master thread (serial code)
Introduction to Parallel Computing
Shared Memory Parallel Programming

OpenMP: An example that does not work!

... lines of codes

!$OMP PARALLEL DO
do k=1,10000

  do j=1,10000
    do i=1,5000
      a(i,j,k)=a(i,j,k+1) - b(j)
      end do
    end do
  end do
end do

!$OMP PARALLEL DO
... lines of codes

One master thread (serial code)
Entering parallel region

Problems!! A thread might use data modified by another thread
Exciting parallel region
One master thread (serial code)
OpenMP: An Example Using Private and Firstprivate Clauses

```plaintext
!$OMP PARALLEL DO PRIVATE(tmp1,tmp2)
!$OMP& FIRSTPRIVATE(b)
do k=1,10000

tmp1=2.34*k*k
  do j=1,10000
    b(j)=b(j)+2
    tmp2 = tmp1 + b(j)
  do i=1,5000
    a(i,j,k)=a(i,j,k) - tmp2
  end do
end do

end do
!$OMP PARALLEL DO
```

Each thread has a copy of tmp1, tmp2 and b

Each thread calculates tmp1 independently

The initial values of b in each thread, is equal to the values of b in the master thread prior to entering the parallel region.
OpenMP: False Sharing!

**False sharing is related to how the system manages the cache in a shared memory environment. To maintain the caches coherency, the system might have to flush a cache line of a CPU associated with data being requested by another CPU.**

```c
$OMP PARALLEL DO
do k=1,10
  do j=1,5
    do i=1,5
      a(i,j,k) = c(i)*k - b(j)
    end do
  end do
end do
$OMP PARALLEL DO
```
1) Distributed Memory Programming Interfaces
2) MPI : Several Instances of a Single Program
3) MPI : A Simple Program to get Started
4) MPI : Communication Models
5) MPI : Domain Decomposition
6) MPI : Commonly Used Calls
7) MPI : Data Types
8) MPI: A Simple Program to Get Started
9) MPI : Point-to-point Communications
10) MPI : Collective Communications
11) MPI : Using Communicators
12) MPI-2 Features
Message Passing Interface (MPI)

- Is a standard in development since 1994
- The most commonly used interface
- Several implementations available
  - MPICH, LAM MPI, Vendors specific
- MPI-1 Standard first release: 1994
- MPI-2 Standard first release: 1996
- MPI Forum (http://www.mpi-forum.org/)
MPI: Several Instances of the Same Program

MPI Parallel program

> Several instances of the same program <

> Each instance represents an MPI Task <

> Usually One Task per CPU or Node <

> A Task does not have direct access to other Tasks memory <

> Communication is done explicitly between Tasks <

> Each TASK has an ID <
Introduction to Parallel Computing
Distributed Memory Parallel Programming

MPI: Several Instances of a Single Program

Memories and Network Switches

Carol Gauthier, Centre de Calcul Scientifique (CCS), Université de Sherbrooke
Master-Slave Model
- Communications between Master task and slave tasks
- The Master dispatches jobs to slaves
- The Master gathers results
- Implies more point-to-point communications than collective
- The Master task and a slave task might run on the same CPU
- Use this model when task jobs are of unpredictable length

Peer-to-peer Model
- No Master or Slave tasks
- Each task knows what it has to do
- Implies more collective communications
- Use this model when task jobs length are predictable
Introduction to Parallel Computing

Distributed Memory Parallel Programming

MPI: Domain Decomposition

1D decomposition

2D decomposition

Initial domain

Carol Gauthier, Centre de Calcul Scientifique (CCS), Université de Sherbrooke
Introduction to Parallel Computing
Distributed Memory Parallel Programming

MPI : Commonly Used Calls

MPI_INIT
MPI_FINILIZE
MPI_COMM_SIZE
MPI_COMM_RANK
MPI_BCAST
MPI_SEND, MPI_ISEND
MPI_RECV, MPI_IRECV
MPI_REDUCE, MPI_ALLREDUCE
MPI_GATHER, MPI_ALLGATHER
MPI_SCATTER

.....

Carol Gauthier, Centre de Calcul Scientifique (CCS), Université de Sherbrooke
# MPI: Data Types

## Basic MPI datatypes for Fortran

<table>
<thead>
<tr>
<th>MPI Datatype</th>
<th>Fortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_BYTE</td>
<td></td>
</tr>
<tr>
<td>MPI_CHARACTER</td>
<td>CHARACTER</td>
</tr>
<tr>
<td>MPI_COMPLEX</td>
<td>COMPLEX*16</td>
</tr>
<tr>
<td>MPI_DOUBLE_PRECISION</td>
<td>REAL*8</td>
</tr>
<tr>
<td>MPI_INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MPI_LOGICAL</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>MPI_REAL</td>
<td>REAL</td>
</tr>
</tbody>
</table>

## Basic MPI datatypes for C

<table>
<thead>
<tr>
<th>MPI Datatype</th>
<th>C Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_BYTE</td>
<td>signed char</td>
</tr>
<tr>
<td>MPI_CHAR</td>
<td>signed char</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>int</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>long</td>
</tr>
<tr>
<td>MPI_LONG_DOUBLE</td>
<td>long double</td>
</tr>
<tr>
<td>MPI_SHORT</td>
<td>short</td>
</tr>
<tr>
<td>MPI_UNSIGNED_CHAR</td>
<td>unsigned char</td>
</tr>
<tr>
<td>MPI_UNSIGNED</td>
<td>unsigned int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_LONG</td>
<td>unsigned long</td>
</tr>
<tr>
<td>MPI_UNSIGNED_SHORT</td>
<td>unsigned short</td>
</tr>
</tbody>
</table>
MPI : A Simple Program to Get Started

```fortran
include "mpif.h"

program my_first_mpi_program

integer taskid,ntasks

call MPI_INIT(error)

call MPI_COMM_RANK(MPI_COMM_WORLD,ntasks)
call MPI_COMM_SIZE(MPI_COMM_WORLD,taskid)

write(6,*)("Hello from task ",{taskid})

call MPI_FINALIZE(error)

end
```

*mpi_init    mpi_finalize    mpi_comm_size    mpi_comm_rank*
**Finite-difference temperature evolution**

- $T(i,j)$: Temperature at current time step
- $Tp(i,j)$: Temperature at previous time step

From Task $k$ point of view:
- $i = 1$ to $n_i$
- $j = k \cdot n_j/n_k + 1$ to $(k+1) \cdot n_j/n_k$

Allocate $T$ and $Tp$ with these limits

We will focus on task $k$ exchanging data ($Tp$) with Task $k-1$ and $k+1$: $Tp$
integer req(4),status(MPI_STATUS_SIZE,4)
call MPI_COMM_SIZE(MPI_COMM_WORLD,k)
call MPI_COMM_RANK(MPI_COMM_WORLD,nk)
jl=k*nj/nk+1
ju=(k+1)*nj/nk
allocate T(1:ni,jl:ju), Tp(1:ni,jl-1:ju+1)
...
call MPI_ISEND(Tp(1,ju),ni,MPI_REAL,k+1,0,
   MPI_COMM_WORLD,req(3),err)
call MPI_IRECV(Tp(1,jl-1),ni,MPI_REAL,k-1,0,
   MPI_COMM_WORLD,req(4),err)
call MPI_ISEND(Tp(1,jl),ni,MPI_REAL,k-1,1,
   MPI_COMM_WORLD,req(1),err)
call MPI_IRECV(Tp(1,ju+1),ni,MPI_REAL,k+1,1,
   MPI_COMM_WORLD,req(2),err)
call MPI_WAITALL(4,req,status,err)

... Calculate T(i,j) with Tp(i,j)
Introduction to Parallel Computing
Distributed Memory Parallel Programming

MPI Collective Communications

Broadcast

Scatter

Gather
**Introduction to Parallel Computing**

**Distributed Memory Parallel Programming**

**MPI Collective Communications**

- **All gather**

  - **Task 0**
    - A
    - B
    - C
    - D

  - **Task 1**
    - A
    - B
    - C
    - D

  - **Task 2**
    - A
    - B
    - C
    - D

  - **Task 3**
    - A
    - B
    - C
    - D

- **All to All**

  - **Task 0**
    - A0
    - A1
    - A2
    - A3

  - **Task 1**
    - B0
    - B1
    - B2
    - B3

  - **Task 2**
    - C0
    - C1
    - C2
    - C3

  - **Task 3**
    - D0
    - D1
    - D2
    - D3

Carol Gauthier, Centre de Calcul Scientifique (CCS), Université de Sherbrooke
Introduction to Parallel Computing
Distributed Memory Parallel Programming

MPI Collective Communications: Reduce and Allreduce

Apply a reduction operation (+, x, logical, etc) to a vector over the set of tasks
Use more than global MPI_COMM_WORLD

Defines groups of tasks

Use Virtual Topologies
MPI-2 : New features

- **Parallel I/O**
- **Remote Memory Operations**
- **Dynamic Process Management**
- **C++ and Fortran 90 bindings**
- **Other improvements**
That's it!

Merci
Thank you

http://www.ccs.usherbrooke.ca