Parallel Programming (Multi/cross-platform)

- **Why Choose C/C++ as the programming language?**
- **Compiling C/C++ on Windows (for free)**
  - Compiling C/C++ on other platforms for free is not an issue
- **Parallel Programming in C/C++ - OpenMP versus MPI**
- **MPI Examples**
- **OpenMP Examples**
- **Project – Assessed Work (50%)**

Why Choose C/C++ as the parallel programming language?

- It is the language most computer science students and scientists are familiar with.

- A large number of parallel algorithm collections/sources contains descriptions (implementations) written in C or in C-like languages.

- C/C++ is closer to the real machine.

- It is nice to use something different from Java.

NOTE: there exists support for parallel programming using a wide range of languages on a variety of OSs.

Compiling C/C++ on Windows (for free)

If you want to compile C/C++ on a windows machine – without paying for it - you have a number of (*reasonable*) choices:

- Microsoft Visual C++ 2010 Express
  Limited libraries (especially for parallel/concurrent development)

- Borland C++ Compiler 5.5
  Limited IDE (basic command line)

- GNU C/C++ Compiler, and associated developement systems:

  - DJGPP – DJ Delorie's GNU Programming Platform
    Limited to executing inside the DOS box as it is a DOS compiler

  - Cygwin – Cygnus's gnuwin32 cross compiler and emulator

  - MinGW32 - Minimalistic GNU for Windows 32 bit

GNU - a recursive acronym for "GNU's Not Unix!"

The most interesting choice is between Cygwin and MinGW32 - where there are trade-offs in terms of portability and licensing.

Cygwin versus MinGW

From Cygwin's web site (http://www.cygwin.com/)

- Cygwin is a Linux-like environment for Windows. It consists of two parts: A DLL (cygwin1.dll) which acts as a Linux API emulation layer providing substantial Linux API functionality.

- A collection of tools which provide Linux look and feel.

- Cygwin uses a DLL, cygwin.dll, (or maybe a set of DLLs) to provide a POSIX-like runtime on Windows. If you build something with Cygwin, any system you install it to will also need the Cygwin DLL(s).

From MinGW's website (http://www.mingw.org/wiki/MinGW)

- MinGW ("Minimalistic GNU for Windows") is a collection of freely available and freely distributable Windows specific header files and import libraries combined with GNU toolsets that allow one to produce native Windows programs that do not rely on any 3rd-party C runtime DLLs

- MinGW compiles to a native Win32 application. A MinGW application does not need any special runtime.

Cygwin versus MinGW

Compile something in Cygwin and you are compiling it *for Cygwin*.

Compile something in MingW and you are compiling it *for Windows*.

Cygwin is good when your app **absolutely needs** a POSIX environment to run - it is sometimes easier to port something to Cygwin than it is to port it to Windows, because Cygwin is a layer on top of Windows that emulates a POSIX environment. If you compile something for Cygwin then it will need to be run within the Cygwin environment, as provided by cygwin1.dll. For portability, you *could* distribute this dll with your project, if you were willing to comply with the relevant license.

MingW is a Windows port of the GNU compiler tools, like GCC, Make, Bash, etc, which run directly in Windows without any emulation layer. By default it will compile to a native Win32 target, complete with .exe and .dll files, though you could also cross-compile with the right settings. It is an alternative to Microsoft Visual C compiler and its associated linking/make tools in a way.

POSIX - Why would we need this in multi-platform development?

POSIX - **P**ortable **O**perating **S**ystem **I**nterface for Uni**x**

A family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces, for software compatible with variants of the Unix operating system, although the standard can apply to any operating system.

POSIX for Windows

- Cygwin - near full compliance *(the most free/open choice)*
- Microsoft POSIX subsystem – partial compliance
- Microsoft Windows Services for UNIX - full POSIX compliance for certain Microsoft Windows products
- UWIN from AT&T Research - near full compliance
- MKS Toolkit - near full compliance

Portability Issues

Case 1: I want to create an application where I write source code once, compile it once and run it in any platforms (e.g. Windows, Linux and Mac OS X…).

Best (?) **Solution** : Perhaps C/C++ is not the best choice. Why not write your source code in JAVA. Compile the source code once and run it anywhere.

Case 2: I want to create an application where I write source code once, and compile the source code separately for each platform.

Best (?) **Solution** : Write your source code in C or C++. Use standard header files only. Use any suitable compiler for any platform. Do not used advanced, machine dependent, language features.

Portability Issues

Case 3: I want to create an application where I write source code once, and compile the source code separately for each platform. However, I need some advanced cross platform features.

Best (?) **Solution** : Use GCC as it is cross platform compiler. On Windows, the simplest solution is **MinGW**. Do not use programming features from the Windows API!

Case 4: I want to create an application where I write source code once, and compile the source code separately for each platform. However, I need advanced features (like mulit-threading) not supported by MinGW.

Best (?) **Solution** : You should use POSIX (Portable Operating System Interface [for UNIX]) standard. It provides many advanced programming features (including multi-threading) and tools. On Windows, **Cygwin** provides a largely POSIX-compliant development and run-time environment.

Distributed versus Shared: MPI versus OpenMP?

Distributed Memory – why MPI?

- Sockets (standardized but low level)
- PVM - Parallel Virtual Machine (obsolete?)
- MPI - Message Passing Interface (de-facto standard)

Shared Memory – why OpenMP?

- Posix Threads (standardized, low level)
- OpenMP – Open Multi-Processing (de-facto standard)
- Automatic Parallelization (compiler does it for you)

ENGINEERING ADVICE: If in doubt - stick to the standards

Distributed versus Shared: MPI versus OpenMP?

Multiple levels of parallelism:

Granularity	*Technology*	*Programming Model*
Instruction Level	*Superscalar*	*Compiler*
Chip Level	*Multicore*	*Compiler, OpenMP, MPI*
System Level	*SMP/cc-NUMA*	*Compiler, OpenMP, MPI*
Grid Level	*Cluster*	*MPI*

Threads Are Getting Cheap

Note: OpenMP and MPI are often combined - in a hybrid system- to provide parallelism at different levels of granularity.

It is often said that MPI is more complicated than MP, but that it scales better… however, this is not accepted by all parallel programmers/engineers

MPI Examples

First check that you have, at the minimum, the following installed:

- mpicc
- mpirun

Check where these are on your file system, and make sure they are in your path. For example, on linux systems you can use the **`which`** command:

```
gibson@gibson-laptop:~/Teaching/OpenMPI$ which mpicc
/usr/bin/mpicc
gibson@gibson-laptop:~/Teaching/OpenMPI$ which mpirun
/usr/bin/mpirun
```

If you wish to compile mpi code using an IDE (like Eclipse) you *may* have to configure the build to be able to find these executables.

Your IDE may provide a specific plug-in to support development in parallel (using MPI and/or other libraries). For example Eclipse has a parallel tools platform at:

http://www.eclipse.org/ptp/downloads.php

mpicc

The **`mpicc`** command can be used to compile and link MPI programs written in C.

It provides the options and any special libraries that are OS dependent

The MPI library may be used with any compiler that uses the same lengths for basic data objects (such as long double) and that uses compatible run-time libraries. On many systems, the various compilers are compatible and may be used interchangably.

The environment variable `MPICH_CC` may be used to select different C compiler and linker. Note that changing the compilers used can cause problems. Use this only if you could intermix code compiled with the different compilers and you know what you are doing.

Combining compilation and linking in a single command is usually done as follows:

```
mpicc -o code code.c
```

The full list of options is found in the man page.

mpirun

mpirun is a shell script which is required to run other mpi based programs.

It attempts to:

- hide the differences in starting jobs for various devices from the user.
- determine what kind of machine it is running on and start the required number of jobs on that machine.

On clusters, you must supply a file that lists the different machines that mpirun can use to run remote jobs or specify this file every time you run mpirun with the -machine file option.

mpirun is typicall invoked as follows:

```
mpirun -np <number of processes> <program name and arguments>
```

If mpirun cannot determine what kind of machine you are on, and it is supported by the mpi implementation, you can use the –machine and -arch options to tell it what kind of machine you are running on

The full list of options is found in the man page.

MPI Examples

HelloWorldMPI.c

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
  int numprocs, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(processor_name, &namelen);

  printf("Hello World, from process %d on %s out of %d\n", rank,
processor_name, numprocs);

  MPI_Finalize();
}
```

MPI Examples

HelloWorldMPI.c

COMPILE

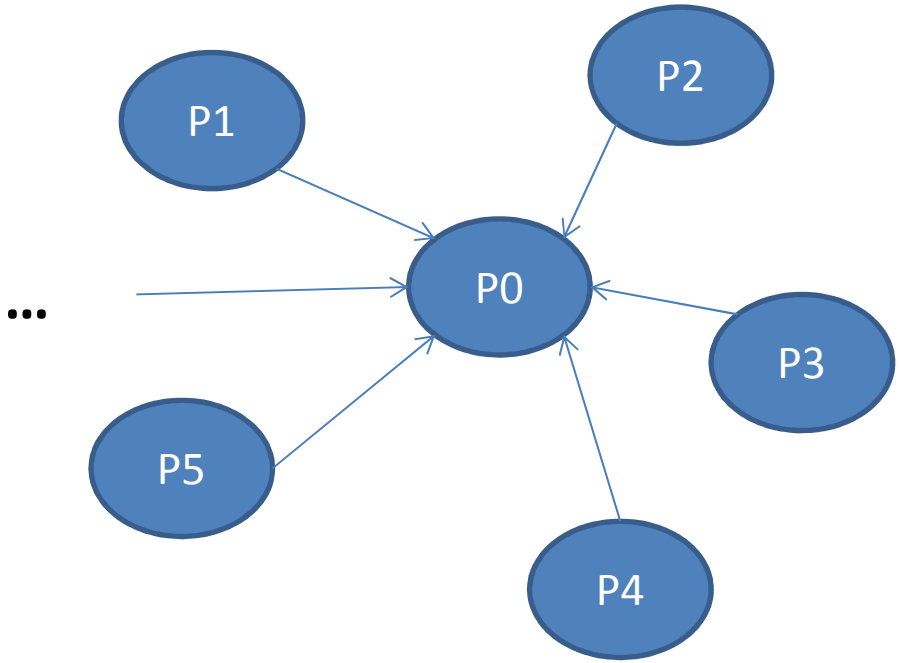
```
gibson@gibson-laptop:~/Teaching/OpenMPI$ mpicc -o HelloWorldMPI
HelloWorldMPI.c
```

EXECUTE ON 4 *PROCESSORS*

```
gibson@gibson-laptop:~/Teaching/OpenMPI$ mpirun -np 4 HelloWorldMPI
Hello World, from process 0 on gibson-laptop out of 4
Hello World, from process 1 on gibson-laptop out of 4
Hello World, from process 2 on gibson-laptop out of 4
Hello World, from process 3 on gibson-laptop out of 4
```

MPI Examples

`testCommStarMPI.c`



MPI Examples

testCommStarMPI.c

COMPILE

```
mpicc -o testCommStarMPI testCommStarMPI.c
```

EXECUTE ON 6 *PROCESSORS*

```
mpirun -np 6 testCommStarMPIProcess 0 will try to receive messages
from 6 processes.
Try to receive message from process 1
Messaged received from process 1 is - Greetings from process 1!
Try to receive message from process 2
Messaged received from process 2 is - Greetings from process 2!
Try to receive message from process 3
Messaged received from process 3 is - Greetings from process 3!
Try to receive message from process 4
Messaged received from process 4 is - Greetings from process 4!
Try to receive message from process 5
Messaged received from process 5 is - Greetings from process 5!
```

MPI Examples

testCommStarMPI.c

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main(int argc, char* argv[]){

        int  my_rank; /* rank of process */
        int  p;       /* number of processes */
        int source;   /* rank of sender */
        int dest;     /* rank of receiver */
        int tag=0;    /* tag for messages */
        char message[100];        /* storage for message */
        MPI_Status status ;   /* return status for receive */

        /* start up MPI */

        MPI_Init(&argc, &argv);

        /* find out process rank */
        MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

        /* find out number of processes */
        MPI_Comm_size(MPI_COMM_WORLD, &p);
```

MPI Examples

testCommStarMPI.c

```
if (my_rank !=0){
                /* create message */
                sprintf(message, "Greetings from process %d!", my_rank);
                dest = 0;
                /* use strlen+1 so that '\0' get transmitted */
                MPI_Send(message, strlen(message)+1, MPI_CHAR,
                   dest, tag, MPI_COMM_WORLD);
        }
else{printf("Process 0 will try to receive messages from %d processes.\n",p);
        for (source = 1; source < p; source++) {
             printf("Try to receive message from process %d\n",source);
              MPI_Recv(message, 100, MPI_CHAR, source, tag,
                                 MPI_COMM_WORLD, &status);
        printf("Messaged received from process %d is - %s\n",source, message);
                }
        }
        /* shut down MPI */
        MPI_Finalize();


        return 0;
}
```

MPI Examples `testCommStarMPI.c`

We could/should restructure the program to separate the **master** and **slave** code:

```
main(){
...
  MPI_Comm_Rank(MPI_COMM_WORLD,&id);
  MPI_Comm_size(MPI_COMM_WORLD,&np);
  if (id ==0 )
            Master();
      else
            Slave();
...
}
```

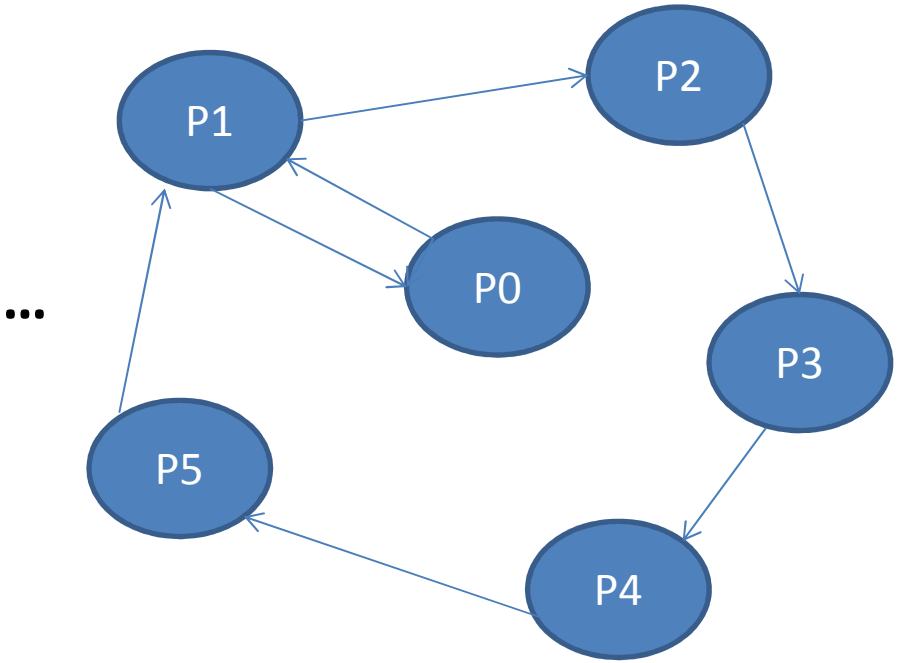
TO DO:
Download the program `testCommStarMPI.c` from the web site

http://www-public.int-evry.fr/~gibson/Teaching/CSC5021/Code/testCommStarMPI.c

and restructure to *master-slave* architecture , as above.

MPI Examples

`testCommRingMPI.c`



TO DO:

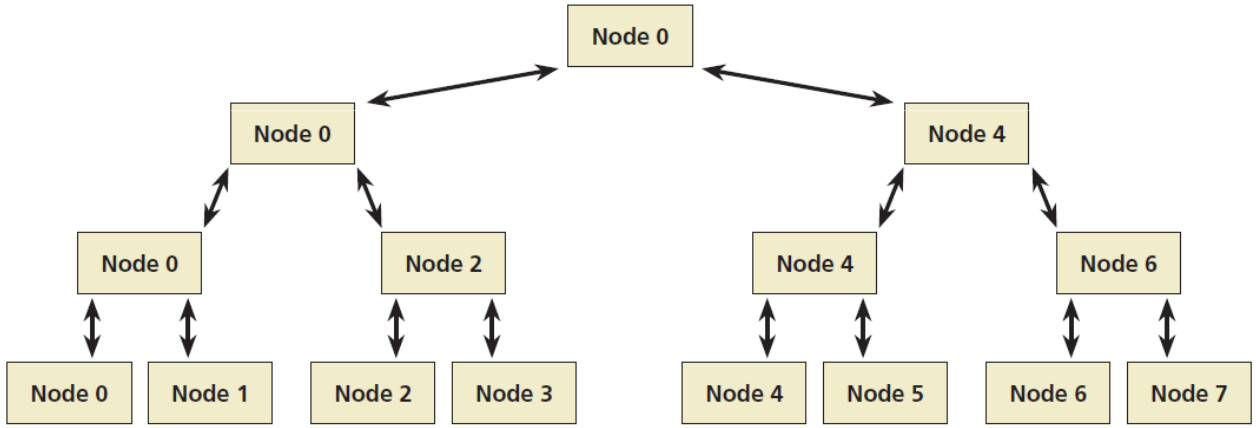
Write a program that establishes a communication ring architecture between processes.

MPI Examples `parallelMergeSortMPI.c`

Original Work – http://penguin.ewu.edu/~trolfe/ParallelMerge/index.html

Timothy J. Rolfe. 2010. A specimen of parallel programming: parallel merge sort implementation. *ACM Inroads* 1, 4 (December 2010), 72-79.

`http://www-public.int-evry.fr/~gibson/Teaching/CSC5021/ReadingMaterial/Rolfe10.pdf`



Process Ranks in a Four-Level Sorting Tree for 8 processes

Download the source code directory **`ParallelMerge.zip`** from the web site:

`http://www-public.int-evry.fr/~gibson/Teaching/CSC5021/Code/ParallelMerge.zip`

MPI Examples

parallelMergeSortMPI.c

```
 mpirun -np 1 parallelMergeSortMPI
1 processes mandates root height of 0
Size:  2671396
Sorting succeeds.
  Parallel:  1.072
Sequential:  1.082
  Speed-up:  1.009
gibson@gibson-laptop:~/Teaching/OpenMPI$ mpirun -np 2 parallelMergeSortMPI
2 processes mandates root height of 1
Size:  14295844
Sorting succeeds.
  Parallel:  6.327
Sequential:  6.177
  Speed-up:  0.976
gibson@gibson-laptop:~/Teaching/OpenMPI$ mpirun -np 4 parallelMergeSortMPI
4 processes mandates root height of 2
Size:  3494692
Sorting succeeds.
  Parallel:  1.553
Sequential:  1.379
  Speed-up:  0.888
gibson@gibson-laptop:~/Teaching/OpenMPI$ mpirun -np 6 parallelMergeSortMPI
6 processes mandates root height of 3
Size:  2949924
Sorting succeeds.
  Parallel:  1.440
Sequential:  1.156
  Speed-up:  0.803
gibson@gibson-laptop:~/Teaching/OpenMPI$ mpirun -np 8 parallelMergeSortMPI
8 processes mandates root height of 3
Size:  16315172
Sorting succeeds.
  Parallel:  8.446
Sequential:  7.444
  Speed-up:  0.881
```

QUESTION: can you explain the speed-up values?

MPI Examples

Checking for prime numbers

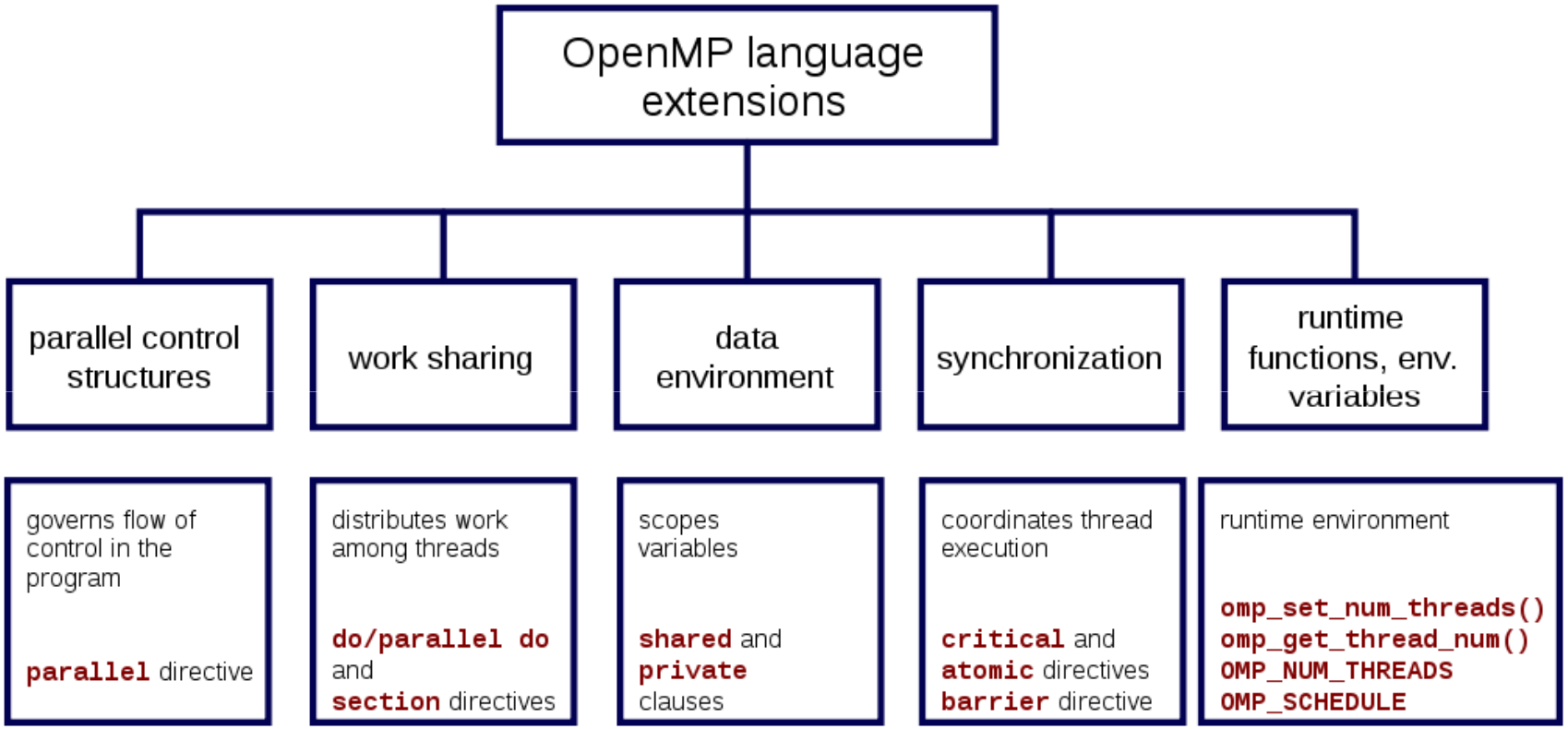
Download the program `parallelprimeCountMPI.c` from the web site:

http://www-public.int-evry.fr/~gibson/Teaching/CSC5021/Code/parallelprimeCountMPI.c

TO DO:

- **Write test code to verify that this functions correctly**
- **Try to understand how the code works**

OpenMP (Open Multi-Processing) is an API (application programming interface) that consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.



In OpenMP with C/C++ the programmer uses #pragmas to control the parallel aspects

OpenMP - resources

The first port of call should be:
http://openmp.org/wp/

There is a good tutorial presentation at
http://www.openmp.org/mp-documents/omp-hands-on-SC08.pdf

A Summary of OpenMP 3.0 C/C++ Syntax can be found at:
http://www.openmp.org/mp-documents/OpenMP3.0-SummarySpec.pdf

There is a good web site on OpenMP with C++ at:
http://bisqwit.iki.fi/story/howto/openmp/#ExampleInitializingATableInParallel

There is a good guide at:
http://geco.mines.edu/workshop/class2/examples/openmp/openmp.pdf

One of the first published articles was:
Leonardo Dagum and Ramesh Menon. 1998. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.* 5, 1 (January 1998), 46-55.
`http://www-public.int-evry.fr/~gibson/Teaching/CSC5021/ReadingMaterial/DagumMenon98.pdf`

OpenMP Examples

HelloWorldOMP.c

```
#include <stdio.h>
#include <omp.h>

int main(int argc, char *argv[]) {
  int iam = 0, np = 1;

  #pragma omp parallel default(shared) private(iam, np)
  {
    #if defined (_OPENMP)
      np = omp_get_num_threads();
      iam = omp_get_thread_num();
    #endif
    printf("Hello from thread %d out of %d\n", iam, np);
  }
}
```

OpenMP Examples

Question: what do you think this code is doing?

```
#pragma omp parallel for
for(int x=0; x < width; x++) {
     for(int y=0; y < height; y++) {
     finalImage[x][y] = RenderPixel(x,y, &sceneData);
     }
}
```

TO DO: Compile and execute some openMP examples. Look for code that does parallel sorting to compare with the MPI merge sort.

Project – Assessed Work (50%)

You are to design, implement, test and compare 2 separate solutions to a **string search problem**:

1. Sequential
2. Parallel

Problem Overview:

Scientists have discovered different types of alien genetic material, sharing common structure:

- 3 dimensional lattice of cubes of distinctive components (letters)
- Each lattice has a finite number of letters in its alphabet – from 2 up to 100
- The lattice sizes range from 4*4*4 up to 1000*1000*1000

Scientists have also discovered dictionaries of words from the genetic alphabet:

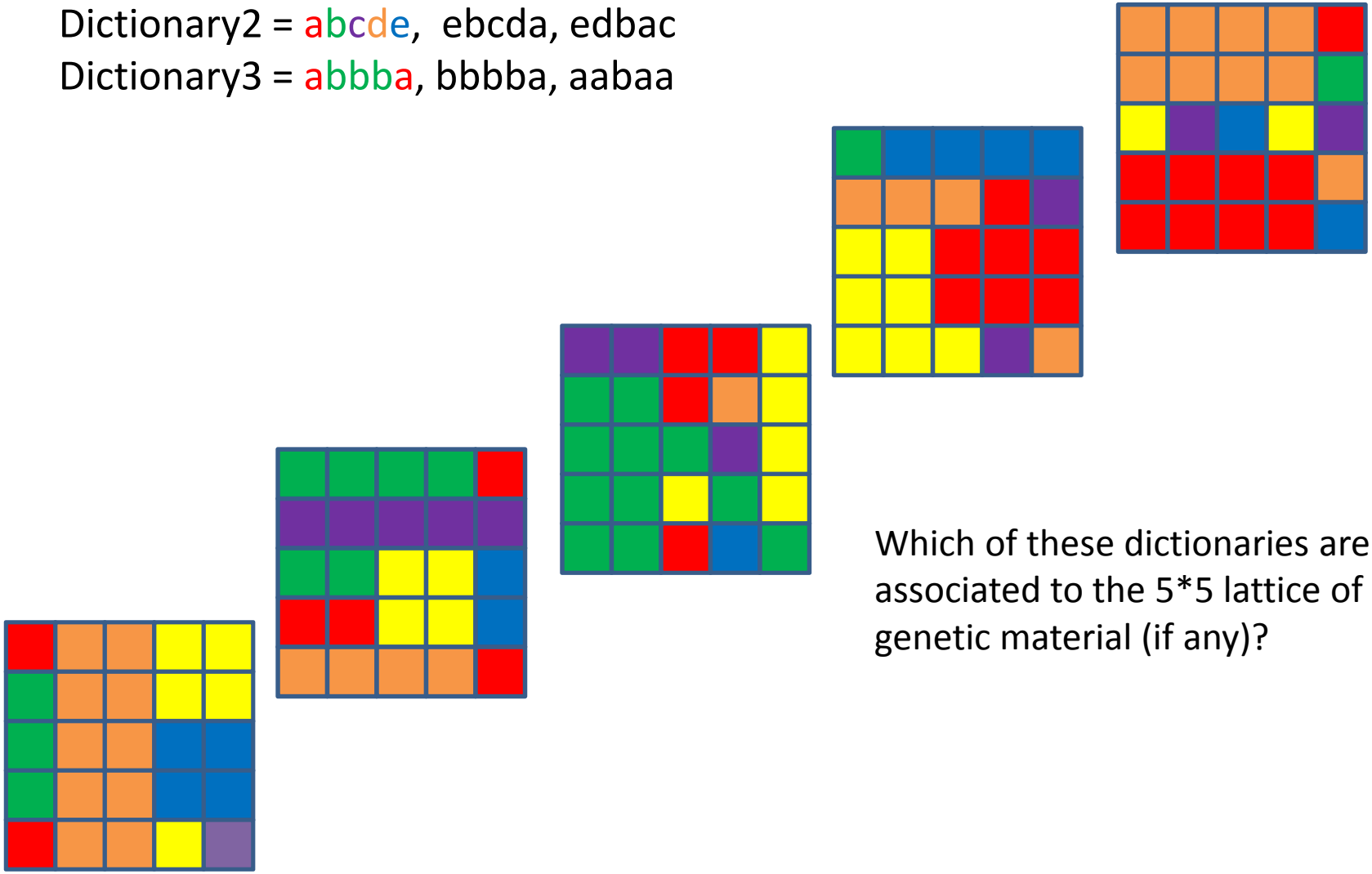
- The dictionary size ranges from 3 up to 1000 words
- All words in a common dictionary share the same length, from 2 up to 100

The scientists hypothesise that dictionaries are associated to some of the genetic material, provided all words in the dictionary can be found in sequence (horizontal, vertical or diagonal) in the material lattice. You are to write a program that takes as input any lattice material and any dictionary, and returns a boolean stating whether there is an association between the two.

Project – Assessed Work (50%)

Example

Dictionary1 = abcb, bcca, aabc
Dictionary2 = abcde, ebcda, edbac
Dictionary3 = abbba, bbbba, aabaa



Which of these dictionaries are associated to the 5*5 lattice of genetic material (if any)?