ECE 462
Object-Oriented Programming using C++ and Java

Parallelism

Yung-Hsiang Lu
yunglu@purdue.edu
Parallelism

- parallel: several activities occurring simultaneously
- Parallelism is natural phenomena in our daily lives
  - Many students take notes in a class.
  - Multiple cars wait for a traffic light.
  - Several lines of people order fast food (and eat).
  - A student uses several washers in a laundry room.
  - Several players try to catch a basketball.
  - A hundred customers order books on-line.
Sequential Programming Model

- One statement executes first, then the next statement.
- Control structures (if, for, while, function call ...) may change the sequence of execution.
- Within a small segment (inside if, for, while, function ...), statements are executed sequentially, one by one.
- Sequential programming model was appropriate when a computer had only one processor.
Parallel Computer on Your Desktop

YHL
Parallel Programming Models

• Multi-threading is one of the most popular, but not the only available programming model.
• Other models include
  – client-server
  – producer-consumer
  – pipeline
  – multithread
  – ...
Client Server

- web browser (client) — web server
- client sends request, server responds
- The client and the server may (in fact, very likely) be multi-thread individually.
Pipeline

- instruction-level parallelism
- factory assembly line
- buffet or BBQ line
Latency and Throughput

Car 1  Car 2  Car 3

S1  S2  S3  S1  S2  S3  S1  S2  S3
divide into stages

S1  S2  S3  S1  S2  S3  S1  S2  S3
pipeline

S3  S1  S2  S3  S1  S2  S3  S1  S2

S2  S3  S1  S2  S3  S1  S2  S3  S1

time

YHL Parallelism 10
Requirements for Parallelism

• multiple processing units
• multiple data
• independence among the units and data
• (usually) some forms of agreements among the units and the data
Multi-Thread
Why is Multiple Thread Important?

• One of the most popular parallel programming models.
• Most machines (through libraries and languages) support multithreads.
• Multiple threads allow a program to respond to different requests quickly.
• You have been using multi-thread programs for years: web browser and most GUI programs.
• It demonstrates many important concepts.
What is a “thread”? 

A thread is (almost) an independent program except
– different threads created in the same program can access shared variables / objects
– smaller overhead compared with processes in context switch
– once a thread starts running, it runs independently until completion or synchronization

context switch = the processor changes which thread to execute

OOP is a natural approach for parallelism. Objects are independent and active.
ECE 462
Object-Oriented Programming using C++ and Java

Threads

Yung-Hsiang Lu
yunglu@purdue.edu
class YourClass extends Thread {
    public void run () {
        // what does this thread do?
    }
}

public static void main(String [] args) {
    YourClass t1 = new YourClass("...");
    t1.start();
}
Qt Thread

#include <QtCore>
class YourClass extends QThread {
    public void run () {
        // what does this thread do?
    }
};
...int main(int argc, char * argv[]) {
    YourClass t1 ("...");
    t1.start();
}
## Thread Execution Time

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t1</th>
<th>t1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td>t2</td>
<td>t2</td>
</tr>
<tr>
<td>t3</td>
<td>t3</td>
<td>t3</td>
<td></td>
<td>t3</td>
</tr>
</tbody>
</table>

YHL Threads 4

time

```
+---+---+---+---+
| t1| t1| t1| t1 |
| t2| t2| t2| t2 |
| t3| t3| t3| t3 |
+---+---+---+---+
```
Many Threads, Few Processors

• advantages of many threads, even on single processor
  – impression of continuous progress in all threads
  – improve utilization of different components
  – handle user inputs more quickly

• disadvantage of many threads
  – add slight work to program structure
  – incur overhead in thread creation
  – cause complex interleaving the execution and possibly wrong results (if you do not think "in parallel")
Java Thread Examples
java.lang

Class Runtime

Java class Runtime extends Object.

Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running. The current runtime can be obtained from the getRuntime method.

An application cannot create its own instance of this class.
<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>addShutdownHook(Thread hook)</td>
<td>Registers a new virtual-machine shutdown hook.</td>
</tr>
<tr>
<td>int</td>
<td>availableProcessors()</td>
<td>Returns the number of processors available to the Java virtual machine.</td>
</tr>
<tr>
<td>Process</td>
<td>exec(String command)</td>
<td>Executes the specified string command in a separate process.</td>
</tr>
<tr>
<td>Process</td>
<td>exec(String[] cmdarray)</td>
<td>Executes the specified command and arguments in a separate process.</td>
</tr>
<tr>
<td>Process</td>
<td>exec(String[] cmdarray, String[] envp)</td>
<td>Executes the specified command and arguments in a separate process.</td>
</tr>
</tbody>
</table>
Matrix Addition

\[
\begin{array}{cccc}
A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & A_{22} & A_{23} & A_{24} \\
A_{31} & A_{32} & A_{33} & A_{34} \\
A_{41} & A_{42} & A_{43} & A_{44} \\
\end{array}
\]

\[
\begin{array}{cccc}
B_{11} & B_{12} & B_{13} & B_{14} \\
B_{21} & B_{22} & B_{23} & B_{24} \\
B_{31} & B_{32} & B_{33} & B_{34} \\
B_{41} & B_{42} & B_{43} & B_{44} \\
\end{array}
\]

\[
\begin{array}{cccc}
C_{11} & C_{12} & C_{13} & C_{14} \\
C_{21} & C_{22} & C_{23} & C_{24} \\
C_{31} & C_{32} & C_{33} & C_{34} \\
C_{41} & C_{42} & C_{43} & C_{44} \\
\end{array}
\]
## Divide Matrix Addition

<table>
<thead>
<tr>
<th>thread 1</th>
<th>thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11</td>
<td>A12</td>
</tr>
<tr>
<td>A21</td>
<td>A22</td>
</tr>
<tr>
<td>A31</td>
<td>A32</td>
</tr>
<tr>
<td>A41</td>
<td>A42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B11</th>
<th>B12</th>
<th>B13</th>
<th>B14</th>
</tr>
</thead>
<tbody>
<tr>
<td>B21</td>
<td>B22</td>
<td>B23</td>
<td>B24</td>
</tr>
<tr>
<td>B31</td>
<td>B32</td>
<td>B33</td>
<td>B34</td>
</tr>
<tr>
<td>B41</td>
<td>B42</td>
<td>B43</td>
<td>B44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C11</th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
</tr>
</thead>
<tbody>
<tr>
<td>C21</td>
<td>C22</td>
<td>C23</td>
<td>C24</td>
</tr>
<tr>
<td>C31</td>
<td>C32</td>
<td>C33</td>
<td>C34</td>
</tr>
<tr>
<td>C41</td>
<td>C42</td>
<td>C43</td>
<td>C44</td>
</tr>
</tbody>
</table>
Matrix Multiplication

\[
\begin{array}{cccc}
A_{11} & A_{12} & A_{13} & A_{14} \\
A_{21} & A_{22} & A_{23} & A_{24} \\
A_{31} & A_{32} & A_{33} & A_{34} \\
A_{41} & A_{42} & A_{43} & A_{44} \\
\end{array}
\]

\[
\begin{array}{cccc}
B_{11} & B_{12} & B_{13} & B_{14} \\
B_{21} & B_{22} & B_{23} & B_{24} \\
B_{31} & B_{32} & B_{33} & B_{34} \\
B_{41} & B_{42} & B_{43} & B_{44} \\
\end{array}
\]

\[
\begin{array}{cccc}
C_{11} & C_{12} & C_{13} & C_{14} \\
C_{21} & C_{22} & \text{C23} & C_{24} \\
C_{31} & C_{32} & C_{33} & C_{34} \\
C_{41} & C_{42} & C_{43} & C_{44} \\
\end{array}
\]
Divide Matrix Multiplication

thread 1

thread 2

<table>
<thead>
<tr>
<th>A11</th>
<th>A12</th>
<th>A13</th>
<th>A14</th>
</tr>
</thead>
<tbody>
<tr>
<td>A21</td>
<td>A22</td>
<td>A23</td>
<td>A24</td>
</tr>
<tr>
<td>A31</td>
<td>A32</td>
<td>A33</td>
<td>A34</td>
</tr>
<tr>
<td>A41</td>
<td>A42</td>
<td>A43</td>
<td>A44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B11</th>
<th>B12</th>
<th>B13</th>
<th>B14</th>
</tr>
</thead>
<tbody>
<tr>
<td>B21</td>
<td>B22</td>
<td>B23</td>
<td>B24</td>
</tr>
<tr>
<td>B31</td>
<td>B32</td>
<td>B33</td>
<td>B34</td>
</tr>
<tr>
<td>B41</td>
<td>B42</td>
<td>B43</td>
<td>B44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C11</th>
<th>C12</th>
<th>C13</th>
<th>C14</th>
</tr>
</thead>
<tbody>
<tr>
<td>C21</td>
<td>C22</td>
<td>C23</td>
<td>C24</td>
</tr>
<tr>
<td>C31</td>
<td>C32</td>
<td>C33</td>
<td>C34</td>
</tr>
<tr>
<td>C41</td>
<td>C42</td>
<td>C43</td>
<td>C44</td>
</tr>
</tbody>
</table>

YHL Threads
main thread

th[0] = new AdderThread ...

th[1] = new AdderThread ...

th[2] = new AdderThread ...

th[0].start();

th[1].start();

th[2].start();

th[0].join();

th[1].join();

th[2].join();
package matrix;

public class MatrixTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Number of Processors = " +
                        Runtime.getRuntime().availableProcessors());

        int row = 1000;
        if (args.length > 0) {
            row = Integer.parseInt(args[0]);
        }

        if (row < 2) {
            row = 2;
        }

        System.out.println("Number of Rows = " + row);
        Matrix ma = new Matrix(row);
        Matrix mb = new Matrix(row);
        Matrix mc = ma.add(mb);
        Matrix md = ma.multiply(mb);
        // ma.print();
        // mb.print();
        // mc.print();
    }
}
public class AdderThread extends Thread {
    private int mt_r1;
    private int mt_r2;
    double[][] mt_Aelements;
    double[][] mt_Belements;
    double[][] mt_Celements;

    public AdderThread(int r1, int r2, double[][] Melements, 
                      double[][] Melements, double[][] Melements) {
        // System.out.println("r1 r2 = " + r1 + " + r2);
        mt_r1 = r1;
        mt_r2 = r2;
        mt_Aelements = Melements;
        mt_Belements = Melements;
        mt_Celements = Melements;
    }

    public void run() {
        int numCol = mt_Aelements[0].length;
        for (int rindex = mt_r1; rindex <= mt_r2; rindex++) {
            for (int cindex = 0; cindex < numCol; cindex++) {
                mt_Celements[rindex][cindex] = mt_Aelements[rindex][cindex] 
                                                + mt_Belements[rindex][cindex];
            }
        }
    }
}
public class MultiplierThread extends Thread{
    private int mt_r1;
    private int mt_r2;
    double[][] mt_Aelements;
    double[][] mt_Belements;
    double[][] mt_Celements;

    public MultiplierThread(int r1, int r2, double[][] MAelements,
                            double[][] MBelements, double[][] MCelements) {
        mt_r1 = r1;
        mt_r2 = r2;
        mt_Aelements = MAelements;
        mt_Belements = MBelements;
        mt_Celements = MCelements;
    }

    public void run() {
        int numCol = mt_Aelements[0].length;
        for (int rindex = mt_r1; rindex <= mt_r2; rindex++) {
            for (int cindex = 0; cindex < numCol; cindex++) {
                mt_Celements[rindex][cindex] = 0;
                for (int eindex = 0; eindex < numCol; eindex++) {
                    mt_Celements[rindex][cindex] += mt_Aelements[rindex][eindex] * mt_Belements[eindex][cindex];
                }
            }
        }
    }
}
package matrix;

import java.io.FileOutputStream;

public class Matrix {
    double[][] m_elements;
    public Matrix(int row, int col) {
        m_elements = new double[row][col];
    }

    public Matrix(int row) {
        m_elements = new double[row][row];
        for (int rindex = 0; rindex < row; rindex++) {
            for (int cindex = 0; cindex < row; cindex++) {
                if (rindex == cindex) {
                    m_elements[rindex][cindex] = 1;
                } else {
                    m_elements[rindex][cindex] = 0;
                }
            }
        }
    }

    public Matrix add(Matrix BM) {
        if ((m_elements.length != BM.m_elements.length)
            || (m_elements[0].length != BM.m_elements[0].length)) {
            System.out.println("add different dimensions");
        }
        System.out.println("Matrix add");
        Matrix CM = new Matrix(m_elements.length, m_elements.length);
        int numProcessor = Runtime.getRuntime().availableProcessors();
        long[] spentTime = new long[2 * numProcessor + 1];
    }
long[] spentTime = new long[2 * numProcessor + 1];
// base line, no threads
long t1 = System.currentTimeMillis();
for (int rindex = 0; rindex < m_elements.length; rindex++) {
    for (int cindex = 0; cindex < m_elements[rindex].length; cindex++) {
        CM.m_elements[rindex][cindex] = m_elements[rindex][cindex] + BM.m_elements[rindex][cindex];
    }
}
long t2 = System.currentTimeMillis();
spentTime[1] = t2 - t1;
for (int pindex = 2; pindex <= numProcessor * 2; pindex++) {
    System.out.println("number of threads = " + pindex);
t1 = System.currentTimeMillis();
AdderThread[] mt = new AdderThread[pindex];
int assignedRow = 0;
int rowPerThread = m_elements.length / pindex;
for (int tindex = 0; tindex < pindex - 1; tindex++) {
    mt[tindex] = new AdderThread(assignedRow, assignedRow + rowPerThread - 1, m_elements, BM.m_elements, CM.m_elements);
    assignedRow += rowPerThread;
}
// assign all remaining rows, this is needed because
// the number of rows may not be a multiple of the number
// of threads
mt[pindex - 1] = new AdderThread(assignedRow,
    m_elements.length - 1, m_elements, BM.m_elements,
public Matrix multiply(Matrix BM) {
    if (m_elements.length != BM.m_elements.length) {
        System.out.println("multiply different dimensions");
    }
    System.out.println("Matrix multiply");
System.out.println("Matrix multiply");
Matrix CM = new Matrix(m_elements[0].length, m_elements.length);
int numProcessor = Runtime.getRuntime().availableProcessors();
long[] spentTime = new long[2 * numProcessor + 1];
// base line, no threads
long t1 = System.currentTimeMillis();
for (int rindex = 0; rindex < m_elements.length; rindex++) {
   for (int cindex = 0; cindex < m_elements[0].length; cindex++) {
      CM.m_elements[rindex][cindex] = 0;
      for (int eindex = 0; eindex < m_elements[0].length; eindex++) {
         CM.m_elements[rindex][cindex] += m_elements[eindex][cindex] * BM.m_elements[eindex][cindex];
      }
   }
}

long t2 = System.currentTimeMillis();
spentTime[0] = t2 - t1;
for (int pindex = 2; pindex <= numProcessor * 2; pindex++) {
   t1 = System.currentTimeMillis();
   MultiplierThread[] mt = new MultiplierThread[pindex];
   int assignedRow = 0;
   int rowPerThread = m_elements.length / pindex;
   for (int tindex = 0; tindex < pindex - 1; tindex++) {
      mt[tindex] = new MultiplierThread(assignedRow, assignedRow + rowPerThread - 1, m_elements, BM.m_elements, CM.m_elements);
      assignedRow += rowPerThread;
   }
for (int pindex = 2; pindex <= numProcessor * 2; pindex++) {
    t1 = System.currentTimeMillis();
    MultiplierThread[] mt = new MultiplierThread[pindex];
    int assignedRow = 0;
    int rowPerThread = m_elements.length / pindex;
    for (int tindex = 0; tindex < pindex - 1; tindex++) {
        mt[tindex] = new MultiplierThread(assignedRow, assignedRow
            + rowPerThread - 1, m_elements, BM.m_elements,
            CM.m_elements);
        assignedRow += rowPerThread;
    }
    mt[pindex - 1] = new MultiplierThread(assignedRow,
        m_elements.length - 1, m_elements, BM.m_elements,
        CM.m_elements);
    for (int tindex = 0; tindex < pindex; tindex++) {
        mt[tindex].start();
    }
    for (int tindex = 0; tindex < pindex; tindex++) {
        try {
            mt[tindex].join();
        } catch (Exception ie) {
        }
    }
    t2 = System.currentTimeMillis();
    spentTime[pindex] = t2 - t1;
}
for (int pindex = 1; pindex <= numProcessor * 2; pindex++) {
try {
    mt[tindex].join();
} catch (Exception ie) {

}

    t2 = System.currentTimeMillis();
    spentTime[pindex] = t2 - t1;

for (int pindex = 1; pindex <= numProcessor * 2; pindex++) {
    System.out.println("thread = " + pindex + " time = "
            + spentTime[pindex]);
}

    return CM;

}

    public void print() {
        System.out.println("row, column = " + m_elements.length + " "+ m_elements[0].length);
        for (int rindex = 0; rindex < m_elements.length; rindex++) {
            for (int cindex = 0; cindex < m_elements[0].length; cindex++) {
                System.out.format("%10.3f", m_elements[rindex][cindex]);
            }
        }

        System.out.println();
    }
package matrix;

public class MatrixTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Number of Processors = " +
                Runtime.getRuntime().availableProcessors());
        int row = 1000;
        if (args.length > 0) {
            row = Integer.parseInt(args[0]);
        }
        if (row < 2) {
            row = 4;
        }
        System.out.println("Number of Rows = " + row);
        Matrix ma = new Matrix(row);
        Matrix mb = new Matrix(row);
        Matrix mc = ma.add(mb);
        Matrix md = ma.multiply(mb);
        // ma.print();
        // mb.print();
        // mc.print();
    }
}
processor      : 0
vendor_id      : GenuineIntel
cpu family     : 6
model          : 15
model name     : Intel(R) Xeon(R) CPU E5310 @ 1.60GHz
stepping       : 7
cpu MHz         : 1595.930
cache size     : 4096 KB
physical id    : 0
siblings       : 4
core id        : 0
cpu cores      : 4
fpu            : yes
fpu_exception  : yes
cpu id level   : 10
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
                mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm syscall
                nx lm monitore ds cpl tm2 cx16 xtrm lahf_lm
bogomips       : 3194.39
cache alignment: 64
address sizes  : 36 bits physical, 48 bits virtual
power management:
<table>
<thead>
<tr>
<th>attribute</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor</td>
<td>7</td>
</tr>
<tr>
<td>vendor_id</td>
<td>GenuineIntel</td>
</tr>
<tr>
<td>cpu family</td>
<td>6</td>
</tr>
<tr>
<td>model</td>
<td>15</td>
</tr>
<tr>
<td>model name</td>
<td>Intel(R) Xeon(R) CPU</td>
</tr>
<tr>
<td>stepping</td>
<td>7</td>
</tr>
<tr>
<td>cpu MHz</td>
<td>1595.930</td>
</tr>
<tr>
<td>cache size</td>
<td>4096 KB</td>
</tr>
<tr>
<td>physical id</td>
<td>1</td>
</tr>
<tr>
<td>siblings</td>
<td>4</td>
</tr>
<tr>
<td>core id</td>
<td>7</td>
</tr>
<tr>
<td>cpu cores</td>
<td>4</td>
</tr>
<tr>
<td>fpu</td>
<td>yes</td>
</tr>
<tr>
<td>fpu_exception</td>
<td>yes</td>
</tr>
<tr>
<td>cpuid level</td>
<td>10</td>
</tr>
<tr>
<td>wp</td>
<td>yes</td>
</tr>
</tbody>
</table>
| flags                | fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm syscall
| bogomips             | 3191.89                     |
| clflush size         | 64                          |
| cache_alignment      | 64                          |
| address sizes        | 36 bits physical, 48 bits virtual |
| power management     |                             |

[(`qstruct04`) ~/ ]
MemTotal: 8165928 kB
MemFree: 7454980 kB
Buffers: 73980 kB
Cached: 448000 kB
SwapCached: 13928 kB
Active: 224628 kB
Inactive: 319472 kB
HighTotal: 0 kB
HighFree: 0 kB
LowTotal: 8165928 kB
LowFree: 7454980 kB
SwapTotal: 8385920 kB
SwapFree: 8365104 kB
Dirty: 4 kB
Writeback: 0 kB
Mapped: 20732 kB
Slab: 138196 kB
CommitLimit: 12468884 kB
Committed_AS: 161364 kB
PageTables: 2512 kB
VmallocTotal: 536870911 kB
VmallocUsed: 267064 kB
VmallocChunk: 536603263 kB
HugePages_Total: 0
Hugepages_Free: 0
--- More --- (0%)
When Java starts, use 1024MB memory. Maximum memory is 2048MB.
thread = 8 time = 8  
thread = 9 time = 8  
thread = 10 time = 9  
thread = 11 time = 10  
thread = 12 time = 9  
thread = 13 time = 9  
thread = 14 time = 9  
thread = 15 time = 10  
thread = 16 time = 9  
Matrix multiply  
thread = 1 time = 39316  
thread = 2 time = 16074  
thread = 3 time = 11006  
thread = 4 time = 4614  
thread = 5 time = 3976  
thread = 6 time = 3375  
thread = 7 time = 2950  
thread = 8 time = 2507  
thread = 9 time = 3294  
thread = 10 time = 2934  
thread = 11 time = 2874  
thread = 12 time = 2572  
thread = 13 time = 2650  
thread = 14 time = 2757  
thread = 15 time = 2594  
thread = 16 time = 2571  
[ (qstruct04) ~/lecturecode/1027/ThreadMatrix/src/ ]
Qt Thread
QThread Class Reference

[QtCore module]

The QThread class provides platform-independent threads. More...

```
#include <QThread>
```

InheritsQObject.

Inherited by
- List of all members, including inherited members
- Qt 3 support members

Public Types
- **QThread** (QObject *parent = 0)
- ~QThread()
- void **exit** (int *returnCode = 0)
- bool **isFinished** () const
- bool **isRunning** () const
- Priority **setPriority** (Priority priority)
- void **setStackSize** (uint stackSize)
- uint **stackSize** () const
- bool **wait** (unsigned long time = ULONG_MAX)

- **29** public functions inherited from QObject

**Public Slots**

- void **quit** ()
- void **start** (Priority priority = InheritPriority)
- void **terminate** ()

- 1 public slot inherited from QObject
void QThread::usleep ( unsigned long usecs )  [static protected]

Causes the current thread to sleep for usecs microseconds.

See also sleep() and msleep().

bool QThread::wait ( unsigned long time - ULONG_MAX )

Blocks the thread until either of these conditions is met:

- The thread associated with this QThread object has finished execution (i.e. when it returns from run()). This function will return true if the thread has finished. It also returns true if the thread has not been started yet.
- time milliseconds has elapsed. If time is ULONG_MAX (the default), then the wait will never timeout (the thread must return from run()). This function will return false if the wait timed out.

This provides similar functionality to the POSIX pthread_join() function.

See also sleep() and terminate().
```cpp
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include "Matrix.h"

using namespace std;

int main(int argc, char * argv[]) {
    cout << "Number of Processors = " << sysconf(_SC_NPROCESSORS_ONLN) << endl;
    int row = 1000;
    if (argc > 1) {
        row = atoi(argv[1]);
    }
    if (row < 2) {
        row = 2;
    }
    cout << "Number of Rows = " << row << endl;
    Matrix ma(row);
    matrix mb(row);
    Matrix mc = ma.add(mb);
    Matrix md = ma.multiply(mb);
    return 0;
}
```
#ifndef MATRIX_H
#define MATRIX_H_

class Matrix {
private:
    int m_row;
    int m_col;
    double * m_elements;
inline int computeIndex(int row, int col) const {
    return row * m_col + col;
}

public:
    Matrix(int row);
    Matrix add(const Matrix & BM) const;
    Matrix multiply(const Matrix & BM) const;
    Matrix(const Matrix & morig);
    matrix & operator = (const matrix & morig);
    virtual ~Matrix();
};

#endif /*MATRIX_H_*/
#ifndef ADDERTHREAD_H
#define ADDERTHREAD_H
#include <QtCore>

class AdderThread : public QThread {
private:
    int at_row;
    int at_col;
    int at_r1;
    int at_r2;
    double * at_aelements;
    double * at_belements;
    double * at_celements;

public:
    AdderThread(int r, int c, int r1, int r2, double * ma, double * mb, double * mc);
    virtual ~AdderThread();
};

#endif /* ADDERTHREAD_H */
```c++
#ifndef MULTIPLIERTHREAD_H
#define MULTIPLIERTHREAD_H
#include <QtCore>

class MultiplierThread : public QThread {
private:
    int mt_row;
    int mt_col;
    int mt_r1;
    int mt_r2;
    double * mt_&elements;
    double * mt_elements;
    double * mt_Celements;

public:
    MultiplierThread(int r, int c, int r1, int r2, double * ma, double * mb, double * mc);
    void run();
    virtual ~MultiplierThread();
};
#endif /*MULTIPLIERTHREAD_H*/
```
#include "AdderThread.h"

AdderThread::AdderThread(int r, int c, int r1, int r2, double * ma, double * mb, double * mc) {
    at_row = r;
    at_col = c;
    at_r1 = r1;
    at_r2 = r2;
    at_elements = ma;
    at_Belements = mb;
    at_Celements = mc;
}

AdderThread::~AdderThread() {
    // do not delete anything since nothing is allocated in the constructor
}

void AdderThread::run() {
    int initIndex = at_r1 * at_col;
    int lastIndex = at_r2 * at_col;
    for (int index = initIndex; index < lastIndex; index++) {
        at_Celements[index] = at_elements[index] + at_Belements[index];
    }
}
#include "Matrix.h"
#include <iostream>
#include <QCoreApplication>
#include "AdderThread.h"
#include "MultiplierThread.h"

using namespace std;

Matrix::Matrix(int row)
{
    m_row = row;
    m_col = row;
    m_elements = new double[row * row];
    if (m_elements == NULL) {
        cout << "element allocation fail" << endl;
    }
    int eindex = 0;
    for (int rindex = 0; rindex < row; rindex++) {
        for (int cindex = 0; cindex < row; cindex++) {
            if (rindex == cindex) {
                m_elements[eindex] = 1;
            } else {
                m_elements[eindex] = 0;
            }
            eindex ++;
        }
    }
}

Matrix::Matrix(const Matrix &morig)
{
    m_row = morig.m_row;
}
Matrix Matrix::add(const Matrix & BM) const {
    if ((m_row != BM.m_row) || (m_col != BM.m_col)) {
        cout << "add different dimensions" << endl;
        return Matrix(1);
    }
    cout << "Matrix add" << endl;
    int numProcessor = sysconf(_SC_NPROCESSORS_ONLN);
    long spentTime[2 * numProcessor + 1];
    Matrix CN(m_row);
    QTime timer;
    timer.start();
    int eindex = 0;
    for (int rindex = 0; rindex < m_row; rindex++) {
        for (int cindex = 0; cindex < m_col; cindex++) {
            CN.m_elements[eindex] = m_elements[eindex] + BM.m_elements[eindex];
            eindex ++;
        }
    }
    spentTime[1] = timer.elapsed();
    for (int pindex = 2; pindex <= numProcessor * 2; pindex++) {
        timer.restart();
        AdderThread * mt = new AdderThread *[pindex];
        int assignedRow = 0;
        int rowPerThread = m_row / pindex;
        for (int tindex = 0; tindex < pindex - 1; tindex++) {
            mt[tindex] = new AdderThread|m_row, m_col,
            assignedRow, assignedRow + rowPerThread - 1, m_elements, BM.m_elements, CN.m_elements, 100);
            mt[tindex].start();
        }
    }
}
spentTime[1] = timer.elapsed();

for (int pindex = 2; pindex <= numProcessor * 2; pindex++) {
    timer.restart();
    AdderThread * mt = new AdderThread [pindex];
    int assignedRow = U;
    int rowPerThread = m_row / pindex;
    for (int tindex = 0; tindex < pindex - 1; tindex++) {
        mt[tindex] = new AdderThread(m_row, m_col,
        assignedRow, assignedRow + rowPerThread - 1, m_elements, BM.m_e
        CM.m_elements);
        assignedRow += rowPerThread;
    }
    mt[pindex - 1] = new AdderThread(m_row, m_col,
    assignedRow, m_row - 1, m_elements, BM.m_elements,
    CM.m_elements);

    for (int tindex = 0; tindex < pindex; tindex++) {
        mt[tindex] -> start();
    }

    for (int tindex = 0; tindex < pindex; tindex++) {
        delete mt[tindex];
    }

    delete [] mt;
    spentTime[pindex] = timer.elapsed();
}
ECE 462
Object-Oriented Programming using C++ and Java

Parallel Programs

Yung-Hsiang Lu
yunglu@purdue.edu
Classification

- S: single, M: multiple, I: instruction, D: data
- SISD: single instruction single data ⇒ sequential program on a single-processor computer
- SIMD: single instruction multiple data ⇒ multiple data elements are operated in the same way, matrix addition
- MISD: multiple instruction single data ⇒ relatively rare, find the eigenvalues and transpose of the same matrix
- MIMD: multiple instruction multiple data ⇒ most general form of parallel computation
Parallelization

• data parallelism $\Rightarrow$ break a big piece of data into smaller units
  – different rows of matrices
  – different bank accounts
  – different books

• instruction parallelism $\Rightarrow$ break a long function into smaller functions
  – different bank transactions
  – different purchase options

• granularity of data and instruction parallelism
Data Sharing

• read only ⇒ no problem

• write ⇒ **BIG** problem, if you are not careful
  – read (d0) before write (d1)
  – write (d0) before read (d1)
  – write (d0) before write (d1)

• will be explained later on the topic of synchronization, mutual exclusion, and critical section
Data Partition

• how to add a group of number?
   ⇒ divide them into pairs and add the pairs recursively

\[ O(\log n) \]
Assumptions

• the time and space (overhead) for thread creation, scheduling, and destruction can be ignored
• number of threads = half the number of operands (initially)
• number of processors = half the number of operands (initially)
• all processors can read the operands within the same amount of time
• the intermediate results are accessible by other processors
Structure of Parallel Programs

- data partition
- create threads
- start thread execution
- collect results

Performance improvement by parallel processing