The goal of this exercise is to learn how to use PISA http://www.tik.ee.ethz.ch/sop/pisa/ which is a multi-objective optimization tool. We will use PISA to solve an optimization problem.

1 Basic structure of PISA

![Diagram showing the architecture of PISA]

PISA is composed of two modules, Selector and Variator (fig:1). These two modules are implemented as separate programs which communicate through text files.

- **Variator** module contains all parts specific to the optimization problem (e.g., evaluation of solutions, problem representation, variation of solutions through crossover and mutation operations).
- **Selector** module contains the parts of an optimization process which are independent of the optimization problem (mainly the selection process). In this exercise, we will be using the IBEA selector. We will not be making any modifications to the selector module. We will be modifying/augmenting the Variator module. The Variator module needs to have the following components:

  1. A proper encoding for the problem.
  2. Generator of initial population.
  3. Implementation of the fitness function.
  4. Implementation of crossover and mutation functions.

2 Our Optimization Problem

What we will try to solve in this exercise is a classical multiprocessor resource matching problem. We have a set of tasks \( T \) and a set of cores \( M \). Depending on where a given task is executed, the computation time of the task is different. Different cores have different power dissipation. We use \( \text{Power}(j) \) to denote the power dissipation of core \( j \). \( \text{Computation}(i, j) \) is the computation time of task \( i \) if it is executed on core \( j \). \( T_j \) denotes the subset of tasks that are executed on core \( j \). The two objectives are the following:

**Objective 1**: Total energy consumption. This can be computed as:

\[
\text{Total Energy} = \sum_{j \in M} \sum_{i \in T_j} \text{Computation}(i, j) \times \text{Power}(j)
\]
Objective 2: Makespan of the task set. This is defined as the time it takes for all tasks to complete execution.

\[
\text{Makespan} = \max_{j \in M} \left( \sum_{i \in T_j} \text{Computation}(i, j) \quad \forall j \in M \right)
\] (2)

In this optimization problem, we are required to find the mapping of tasks to cores (determine subsets \( T_j \forall j \in M \)) such the two objectives are minimized. We will now iterate through the various steps which need to be followed to solve this problem.

3 Downloading and Understanding PISA

We have prepared the basic problem formulation which you will build on through the course of this exercise. Download and extract PISA and the basic problem formulation using the following commands:

\[
\begin{align*}
\text{wget} & \quad \text{http://www.tik.ee.ethz.ch/education/lectures/hswcd/exercises/EMO_EXERCISE.tar.bz2} \\
\text{tar} & \quad -xvjf \text{EMO_EXERCISE.tar.bz2} \\
\text{cd} & \quad \text{EMO_EXERCISE}
\end{align*}
\]

In this exercise, we will be using a PISA’s Matlab interface. Execute the following command to run Matlab:

```
matlab
```

You will see several files in the extracted emoExercise folder.

*pisa*.m files (pisaConnect.m, pisaDisConnect.m, PISA_ini, pisaInitialize.m, pisaReset.m, pisaRun.m): These files are specific for the Selector Module. You Do Not have to worry about the contents of these files. The focus of this exercise is the Variator module.

In the “Specification/” folder, we have several text files which state the computation times of tasks and power dissipation of cores. We will be reading these text files to generate optimization problem specification.

*emo*.m files (emoCrossover.m, emoExercise.m, emoFitness.m, emoInitial.m, emoMutate.m, emoSpecification.m). These files are specific for the Variator module and we will be augmenting these files over the course of this exercise. Following is a brief explanation of what each of these files do:

(a) emoExercise.m: Matlab script that gels everything together. Its functions include: configuring the optimization parameters, providing a communication interface between selector and variator, running the optimization problem and, displaying/plotting results.

(b) emoCrossover.m: Matlab function that takes as an input two solutions and performs crossover operation on them.

(c) emoFitness.m: Matlab function that evaluates the fitness of a given solution. In the context of our problem, this function computes the energy consumption and makespan of a given solution.

(d) emoInitial.m: Generates initial population of random solutions.

(e) emoMutate.m: Matlab function that mutates a given solution. Currently, this function just returns the input argument itself without performing any mutation. Designing a “good” mutation function is one of your tasks in this exercise.

(f) emoSpecification.m: Matlab function that generates the problem specification. This function reads the text files in “Specification/” folder.

Note: At any point, if you terminate emoExercise.m (using Ctrl-C), or receive a compilation error, call pisaDisConnect() before re-running emoExercise.m. This function removes all temporary files and calling it is essential for proper selector-variator communication.

We can now start solving some optimization problems!!
6.1 Uniform multi-core with DVFS capability

In this task, we assume that M has four cores with following frequency and power dissipation values:

<table>
<thead>
<tr>
<th>Core</th>
<th>Frequency (Ghz)</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 1</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>Core 2</td>
<td>1.5</td>
<td>9</td>
</tr>
<tr>
<td>Core 3</td>
<td>2.0</td>
<td>16</td>
</tr>
<tr>
<td>Core 4</td>
<td>2.5</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 1: DVFS multi-core power dissipation

Notice that power dissipation is proportional to the square of frequency. This is typical for platforms that are capable of changing both voltage and frequency to reduce power dissipation. The computation times of tasks are inversely proportional to frequency. We will now try to solve the optimization problem for this platform.

(a) Can you deduce any properties of the two extreme pareto optimal solutions (solution which has the minimum energy and solution which has the minimum makespan)?

(b) In emoSpecification.m, un-comment the following lines of code (Note: These lines may be un-commented already):

```
1  runtime=dlmread('Specification/Runtime');
2  power=dlmread('Specification/Power_DVFS');
```

With this code uncommented, the power dissipation numbers are as specified in Table 1 and all computation times are inversely proportional to operating frequencies. Run emoExercise.m to solve the optimization problem. Analyze the results and compare the extreme pareto optimal solutions with what you expected in part (a). Explain in case you do not get good solutions. (Hint: Look at how initial population is generated in emoInitial.m)

(c) Implement a mutation operation in emoMutate.m. Rerun emoExercise.m to see if solutions improve. Compare the extreme pareto optimal solutions with what you expected in part (a).

6.2 Uniform multi-core with DVFS and un-core power

In a computing system, there are several non-core components that dissipate power. These include buses, shared memory, etc.

(a) Change the problem formulation such that there is an additional power component $P_{\text{uncore}} = 10W$. This power is dissipated if at least one core is active. Now the energy consumption equation changes to:

```
\text{Total\,Energy} = \sum_{j \in M} \sum_{i \in T_j} \left( \text{Computation}(i, j) \times \text{Power}(j) \right) + P_{\text{uncore}} \cdot \max_{j \in M} \sum_{i \in T_j} \text{Computation}(i, j)
```

(b) Run emoExercise.m to solve the optimization problem. Compare the solutions with the solutions of 6.1(c).

6.3 Uniform multi-core with DFS

For this task, undo the changes you made in the previous task or set $P_{\text{uncore}} = 0$. Re-comment the lines of code you un-commented in part 6.1 (b).

Now we consider a multi-core where the power dissipation is directly proportional to the frequency, instead being directly proportional to the square of frequency. This is typical for platforms that are able to change frequency and are unable to change voltage to reduce power dissipation. The computation times of tasks are inversely proportional to frequency. In this case, we have the following core frequencies and power dissipation:

(a) Can you deduce any properties of the two extreme pareto optimal solutions (solution which has the minimum energy and solution which has the minimum makespan)?
<table>
<thead>
<tr>
<th>Core 1</th>
<th>1.0</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 2</td>
<td>1.5</td>
<td>15</td>
</tr>
<tr>
<td>Core 3</td>
<td>2.0</td>
<td>20</td>
</tr>
<tr>
<td>Core 4</td>
<td>2.5</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 2: DFS multi-core power dissipation

(b) In emoSpecification.m, un-comment the following lines of code:

1. `runtime=dlmread('Specification/Runtime');`
2. `power=dlmread('Specification/Power_DFS');`

Run emoExercise.m to solve the optimization problem. Analyze the results and compare the extreme pareto optimal solutions with what you expected in part (a).

Re-comment the lines of code you un-commented in part (b)

6.4 Heterogeneous multi-core

Re-comment the lines of code you un-commented in part 6.3 (b). Now we consider a heterogeneous multi-core. Core power dissipations are as in Task 6.1. Computation times are arbitrary (they are different depending on the core where a given task is executed but are not dependent on frequency or power dissipation).

(a) In emoSpecification.m, un-comment the following lines of code:

1. `runtime=dlmread('Specification/Runtime_Hetro');`
2. `power=dlmread('Specification/Power_DVFS');`

Run emoExercise.m to solve the optimization problem. Comment on the results.

(b) (Bonus Question) Implement a penalty function such that, in the set of pareto optimal solutions, the number of tasks assigned to any given core is $\leq 45$. 