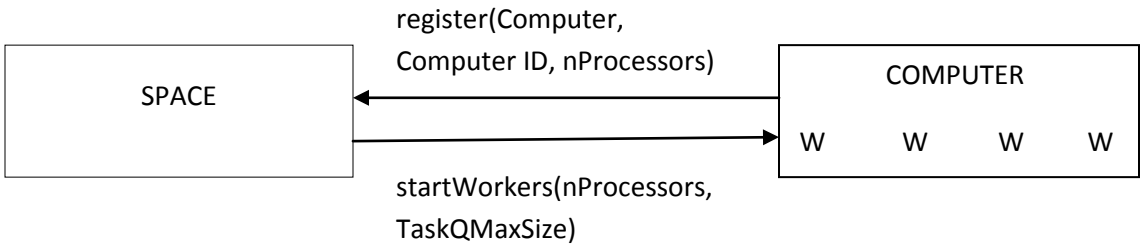
LEVERAGING MULTICORE PROCESSORS:

When a *Computer* registers with the *Space,* it communicates to the *Space* the number of processors available to the JVM. The *Space* then instantiates as many *Worker* threads as the number of processors in the *Computer* to increase the available computational power.



AMELIORATING THE COMMUNICATION LATENCY BETWEEN *SPACE* AND *COMPUTER*:

To reduce the communication latency between the *Space* and *Computers*, the following strategies are followed :

1) **Local Task Queue and *ResultSink* :**
   Every *Computer* maintains a local queue of tasks. Every *Worker* thread in the computer fetches tasks from this queue and executes them. The results of the execution are written into a *ResultSink* Locally. The purpose of *ResultSink* is to overcome the overhead of communicating the results to the *Space* via the *Worker* thread. *ResultSink* is designed to work as a separate thread that periodically writes all the results in its sink to the *Space*.

2) **Task Caching**
   The execution of any *Divide & Conquer* task generates new sub-tasks. Before sending these subtasks to *Space,* an attempt is made to push a portion of the sub-tasks into the local task queue of the *Computer.* This avoids the RMI involved in the *Computer* sending the tasks to the *Space* and receiving them again from the *Space* for execution.

3) ***SpaceRunnable* Tasks**
   Some tasks which are not computationally intensive like the base case of a *TSP* or *Fibonacci* problem can be run on *Space* without the need to pass them to a remote *Computer*.
   To aid this processing, there exists an instance of *Computer* on the *Space* as well.
   Every task that the client wants to execute on *Space* should implement a *SpaceRunnable* marker interface. Before sending a task to a *Computer, Space* checks if the task is instanceof *SpaceRunnable* and queues it in the local *Computer* accordingly.

4) Fault Tolerance

Every *ComputerProxy* object in the *Space* maintains the current list of tasks that are currently part of the local queue of the remote *Computer* corresponding to the proxy. This way, even if one of the compute nodes fail at run-time, tasks that belong to the failed node can be retrieved and reassigned to a different node.

5) Pushing of tasks from *Space*

The *Space*'s main thread continuously monitors the size of the task queue of every *Computer* and pushes tasks to *Computers* whenever the number of tasks in their queues decrease below a certain threshold. This prevents *Computers* from being idle until new tasks are received.