|  |  |  |
| --- | --- | --- |
| 1. | True/False | |
|  | Q: | T/F: Pipelining facilitates “functional decomposition.” |
|  |  |  |
|  | A: | True, pipelining strategies often have computing nodes all perform their own function on the problem data. |
|  |  |  |
|  | Ref: | Page 140 |

|  |  |  |
| --- | --- | --- |
| 2. | Multiple Choice | |
|  | Q: | Which of the following is not a type of problem that would be ideally considered for “pipelining”:   1. A series of data items must be processed, each requiring multiple operations 2. Information to start a next process can be passed forward before the current process completes all of its operations 3. The problem requires several types of functions to be solved 4. More than one instance of the complete problem is to be executed |
|  |  |  |
|  | A: | C. This option is made up and very vague. The other problem types are all discussed in the text. |
|  |  |  |
|  | Ref: | Page 142 |

|  |  |  |
| --- | --- | --- |
| 3. | Fill in the Blank | |
|  | Q: | A \_\_\_\_\_\_\_\_\_ blocking send() is good to use in pipelining schemes because the process is allowed to work on its next item after sending a completed one. |
|  |  |  |
|  | A: | locally. Synonyms are acceptable. The locally blocking send, once again, proves to be superior. |
|  |  |  |
|  | Ref: | Page 145 |

|  |  |  |
| --- | --- | --- |
| 4. | Short Answer/Code | |
|  | Q: | The naïve pipelining approach for adding numbers specifies performing one arithmetic operation, and then passing the result on to the next process. Briefly discuss how this approach can be improved. |
|  |  |  |
|  | A: | This approach can be improved through a partitioning strategy. For example, having the first process adding the first *n/p* numbers in a set, then sending that result on would cut down on communication and make the algorithm more scalable. |
|  |  |  |
|  | Ref: | Page 146 |