



COMP SUPERSCALAR

COMPSs Sample Applications

VERSION: 1.3

August 17, 2015



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

This manual only provides Java, Python and C/C++ sample applications that can be used as a reference. Specifically, it details the applications' code and different ways to execute and analyse them (by using different runcompss flags).

For further information about the application execution please refer to the *COMPSs User Manual: Application execution guide* available at <http://compss.bsc.es> .

For further information about the application development please refer to the *COMPSs User Manual: Application development guide* available at <http://compss.bsc.es/> .

Contents

1	COMP Superscalar (COMPSs)	1
2	Java Sample applications	2
2.1	Matrix multiplication	2
2.2	Sparse LU decomposition	3
2.3	BLAST Workflow	4
3	Python Sample applications	6
3.1	Matrix multiplication	6
3.2	Sparse LU decomposition	7
3.3	BLAST Workflow	8
4	C/C++ Sample applications	10
4.1	Matrix multiplication	10
4.2	Sparse LU decomposition	11
4.3	BLAST Workflow	12

List of Figures

1	The COMPSs Blast workflow.	4
2	The COMPSs Blast workflow.	8
3	The COMPSs Blast workflow.	12

List of Tables

1 COMP Superscalar (COMPSs)

COMP Superscalar (COMPSs) is a programming model which aims to ease the development of applications for distributed infrastructures, such as Clusters, Grids and Clouds. COMP Superscalar also features a runtime system that exploits the inherent parallelism of applications at execution time.

For the sake of programming productivity, the COMPSs model has four key characteristics:

- **Sequential programming:** COMPSs programmers do not need to deal with the typical duties of parallelization and distribution, such as thread creation and synchronization, data distribution, messaging or fault tolerance. Instead, the model is based on sequential programming, which makes it appealing to users that either lack parallel programming expertise or are looking for better programmability.
- **Infrastructure unaware:** COMPSs offers a model that abstracts the application from the underlying distributed infrastructure. Hence, COMPSs programs do not include any detail that could tie them to a particular platform, like deployment or resource management. This makes applications portable between infrastructures with diverse characteristics.
- **Standard programming languages:** COMPSs is based on the popular programming language Java, but also offers language bindings for Python and C/C++ applications. This facilitates the learning of the model, since programmers can reuse most of their previous knowledge.
- **No APIs:** In the case of COMPSs applications in Java, the model does not require to use any special API call, pragma or construct in the application; everything is pure standard Java syntax and libraries. With regard the Python and C/C++ bindings, a small set of API calls should be used on the COMPSs applications.

2 Java Sample applications

The examples in this section consider the execution in a cloud platform where the VMs mount a common storage on **/sharedDisk** directory. This is useful in the case of applications that require working with big files, allowing to transfer data only once, at the beginning of the execution, and to enable the application to access the data directly during the rest of the execution.

The Blast sample workflows available in the development environment and explained in the next section takes advantage of this functionality.

The development environment provides some sample COMPSs applications that can be found in **/home/user/workspace/** directory. The following section describes in detail the development of each of them.

2.1 Matrix multiplication

Matrix Multiplication (Matmul) is a pure Java application that multiplies two matrices in a direct way. The application creates 2 matrices of $N \times N$ size initialized with values, and multiply the matrices by blocks of 40 floats (by default).



`./Sections/4_Sample_Apps/Figures/matrix.jpeg`

In this application the multiplication is implemented in the `multiplyAccumulative` that

is thus selected as the task that will be executed remotely. In order to run the application the matrix dimension has to be supplied.

```
user@bsccompss:~$ cp ~/workspace/matmul/package/Matmul.tar.gz /home/user/  
user@bsccompss:~$ tar xzf Matmul.tar.gz  
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/matmul.jar
```

The command line to execute the application:

```
user@bsccompss:~$ runcompss matmul.Matmul <matrix_dim>
```

2.2 Sparse LU decomposition

SparseLU multiplies two matrices using the factorization method of LU decomposition, which factorizes a matrix as a product of a lower triangular matrix and an upper one.



./Sections/4_Sample_Apps/Figures/SparseLU.jpeg

The matrix is divided into $N \times N$ blocks on where 4 types of operations will be applied modifying the blocks: **lu0**, **fwd**, **bdiv** and **bmod**. These four operations are implemented in four methods that are selected as the tasks that will be executed remotely. In order to run the application the matrix dimension has to be provided.

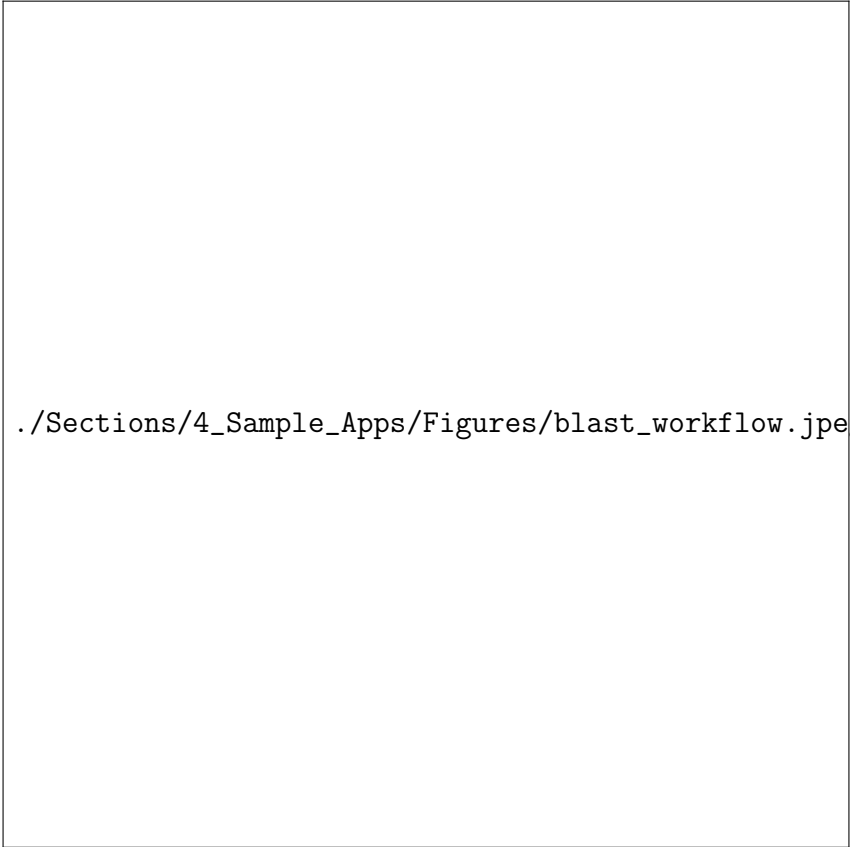
```
user@bsccompss:~$ cp ~/workspace/sparselu/package/SparseLU.tar.gz /home/  
user/  
user@bsccompss:~$ tar xzf SparseLU.tar.gz  
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/sparselu.jar
```

The command line to execute the application:

```
user@bsccompss:~$ runcompss sparselu.SparseLU <matrix_dim>
```

2.3 BLAST Workflow

BLAST is a widely-used bioinformatics tool for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences with sequence databases, identifying sequences that resemble the query sequence above a certain threshold. The work performed by the COMPSs Blast workflow is computationally intensive and embarrassingly parallel.



./Sections/4_Sample_Apps/Figures/blast_workflow.jpeg

Figure 1: The COMPSs Blast workflow.

The workflow describes the three blocks of the workflow implemented in the **Split**, **Align** and **Assembly** methods. The second one is the only method that is chosen to be executed remotely, so it is the unique method defined in the interface file. The **Split** method chops the query sequences file in N fragments, **Align** compares each sequence fragment against the database by means of the Blast binary, and **Assembly** combines all intermediate files into a single result file.

This application uses a database that will be on the shared disk space avoiding transferring the entire database (which can be large) between the virtual machines.

```
user@bsccompss:~$ cp ~/workspace/blast/package/Blast.tar.gz /home/user
/
user@bsccompss:~$ tar xzf Blast.tar.gz
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/blast.jar
```

The command line to execute the workflow:

```
user@bsccompss:~$ runcompss blast.Blast <debug> <bin_location>
                                <database_file>
                                <sequences_file>
                                <frag_number> <tmpdir>
                                <output_file>
```

Where:

- **debug**: The debug flag of the application (true or false).
- **bin_location**: Path of the Blast binary.
- **database_file**: Path of database file; the shared disk **/sharedDisk/** is suggested to avoid big data transfers.
- **sequences_file**: Path of sequences file.
- **frag_number**: Number of fragments of the original sequence file, this number determines the number of parallel Align tasks.
- **tmpdir**: Temporary directory (**/home/user/tmp/**).
- **output_file**: Path of the result file.

Example:

```
user@bsccompss:~$ runcompss blast.Blast true
/home/user/workspace/blast/binary/blastall
/sharedDisk/Blast/databases/swissprot/swissprot
/sharedDisk/Blast/sequences/sargasso_test.fasta 4 /tmp/
/home/user/out.txt
```

3 Python Sample applications

The examples in this section consider the execution in a cloud platform where the VMs mount a common storage on **/sharedDisk** directory. This is useful in the case of applications that require working with big files, allowing to transfer data only once, at the beginning of the execution, and to enable the application to access the data directly during the rest of the execution.

The Blast sample workflows available in the development environment and explained in the next section takes advantage of this functionality.

The development environment provides some sample COMPSs applications that can be found in **/home/user/workspace/** directory. The following section describes in detail the development of each of them.

3.1 Matrix multiplication

Matrix Multiplication (Matmul) is a pure Java application that multiplies two matrices in a direct way. The application creates 2 matrices of $N \times N$ size initialized with values, and multiply the matrices by blocks of 40 floats (by default).



`./Sections/4_Sample_Apps/Figures/matrix.jpeg`

In this application the multiplication is implemented in the `multiplyAccumulative` that

is thus selected as the task that will be executed remotely. In order to run the application the matrix dimension has to be supplied.

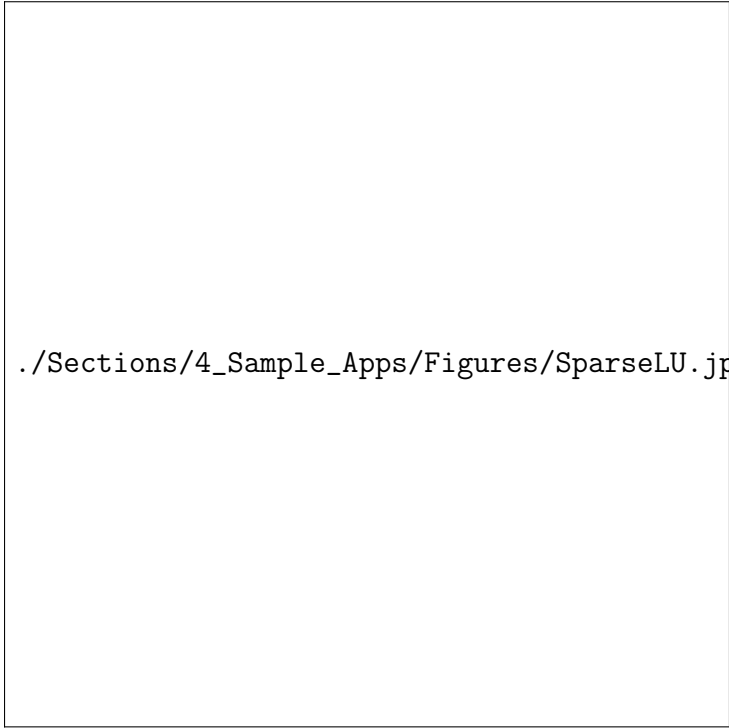
```
user@bsccompss:~$ cp ~/workspace/matmul/package/Matmul.tar.gz /home/user/  
user@bsccompss:~$ tar xzf Matmul.tar.gz  
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/matmul.jar
```

The command line to execute the application:

```
user@bsccompss:~$ runcompss matmul.Matmul <matrix_dim>
```

3.2 Sparse LU decomposition

SparseLU multiplies two matrices using the factorization method of LU decomposition, which factorizes a matrix as a product of a lower triangular matrix and an upper one.



./Sections/4_Sample_Apps/Figures/SparseLU.jpeg

The matrix is divided into $N \times N$ blocks on where 4 types of operations will be applied modifying the blocks: **lu0**, **fwd**, **bdiv** and **bmod**. These four operations are implemented in four methods that are selected as the tasks that will be executed remotely. In order to run the application the matrix dimension has to be provided.

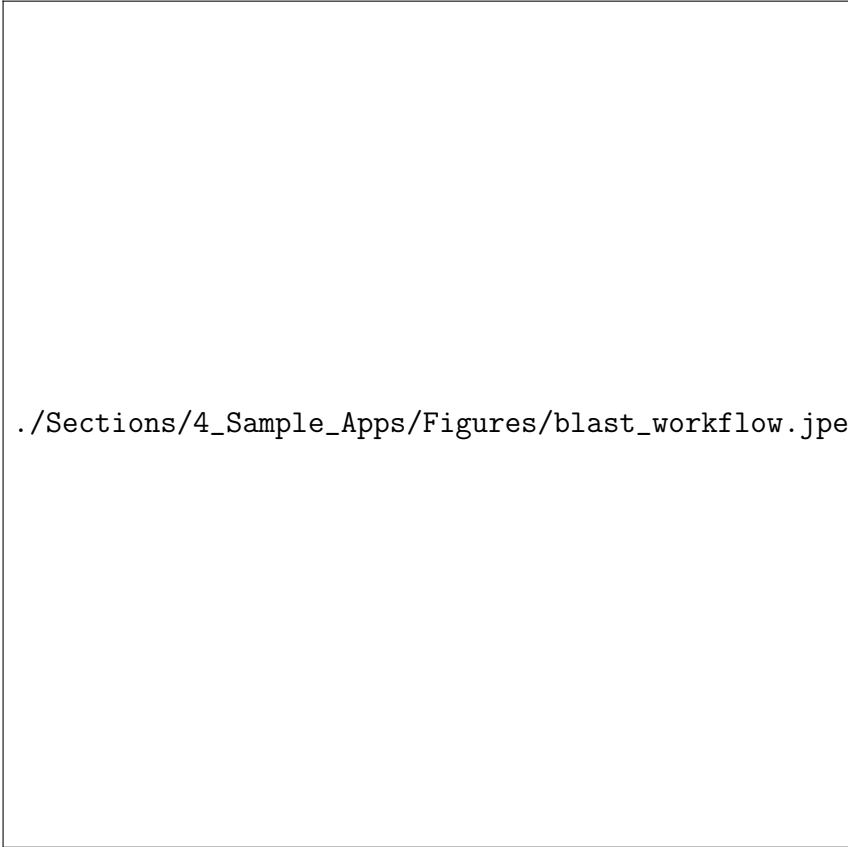
```
user@bsccompss:~$ cp ~/workspace/sparselu/package/SparseLU.tar.gz /home/  
user/  
user@bsccompss:~$ tar xzf SparseLU.tar.gz  
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/sparselu.jar
```

The command line to execute the application:

```
user@bsccompss:~$ runcompss sparselu.SparseLU <matrix_dim>
```

3.3 BLAST Workflow

BLAST is a widely-used bioinformatics tool for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences with sequence databases, identifying sequences that resemble the query sequence above a certain threshold. The work performed by the COMPSs Blast workflow is computationally intensive and embarrassingly parallel.



./Sections/4_Sample_Apps/Figures/blast_workflow.jpeg

Figure 2: The COMPSs Blast workflow.

The workflow describes the three blocks of the workflow implemented in the **Split**, **Align** and **Assembly** methods. The second one is the only method that is chosen to be executed remotely, so it is the unique method defined in the interface file. The **Split** method chops the query sequences file in N fragments, **Align** compares each sequence fragment against the database by means of the Blast binary, and **Assembly** combines all intermediate files into a single result file.

This application uses a database that will be on the shared disk space avoiding transferring the entire database (which can be large) between the virtual machines.

```
user@bsccompss:~$ cp ~/workspace/blast/package/Blast.tar.gz /home/user
/
user@bsccompss:~$ tar xzf Blast.tar.gz
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/blast.jar
```

The command line to execute the workflow:

```
user@bsccompss:~$ runcompss blast.Blast <debug> <bin_location>
                                <database_file>
                                <sequences_file>
                                <frag_number> <tmpdir>
                                <output_file>
```

Where:

- **debug**: The debug flag of the application (true or false).
- **bin_location**: Path of the Blast binary.
- **database_file**: Path of database file; the shared disk **/sharedDisk/** is suggested to avoid big data transfers.
- **sequences_file**: Path of sequences file.
- **frag_number**: Number of fragments of the original sequence file, this number determines the number of parallel Align tasks.
- **tmpdir**: Temporary directory (**/home/user/tmp/**).
- **output_file**: Path of the result file.

Example:

```
user@bsccompss:~$ runcompss blast.Blast true
/home/user/workspace/blast/binary/blastall
/sharedDisk/Blast/databases/swissprot/swissprot
/sharedDisk/Blast/sequences/sargasso_test.fasta 4 /tmp/
/home/user/out.txt
```

4 C/C++ Sample applications

The examples in this section consider the execution in a cloud platform where the VMs mount a common storage on **/sharedDisk** directory. This is useful in the case of applications that require working with big files, allowing to transfer data only once, at the beginning of the execution, and to enable the application to access the data directly during the rest of the execution.

The Blast sample workflows available in the development environment and explained in the next section takes advantage of this functionality.

The development environment provides some sample COMPSs applications that can be found in **/home/user/workspace/** directory. The following section describes in detail the development of each of them.

4.1 Matrix multiplication

Matrix Multiplication (Matmul) is a pure Java application that multiplies two matrices in a direct way. The application creates 2 matrices of N x N size initialized with values, and multiply the matrices by blocks of 40 floats (by default).



`./Sections/4_Sample_Apps/Figures/matrix.jpeg`

In this application the multiplication is implemented in the `multiplyAccumulative` that

is thus selected as the task that will be executed remotely. In order to run the application the matrix dimension has to be supplied.

```
user@bsccompss:~$ cp ~/workspace/matmul/package/Matmul.tar.gz /home/user/  
user@bsccompss:~$ tar xzf Matmul.tar.gz  
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/matmul.jar
```

The command line to execute the application:

```
user@bsccompss:~$ runcompss matmul.Matmul <matrix_dim>
```

4.2 Sparse LU decomposition

SparseLU multiplies two matrices using the factorization method of LU decomposition, which factorizes a matrix as a product of a lower triangular matrix and an upper one.



./Sections/4_Sample_Apps/Figures/SparseLU.jpeg

The matrix is divided into $N \times N$ blocks on where 4 types of operations will be applied modifying the blocks: **lu0**, **fwd**, **bdiv** and **bmod**. These four operations are implemented in four methods that are selected as the tasks that will be executed remotely. In order to run the application the matrix dimension has to be provided.

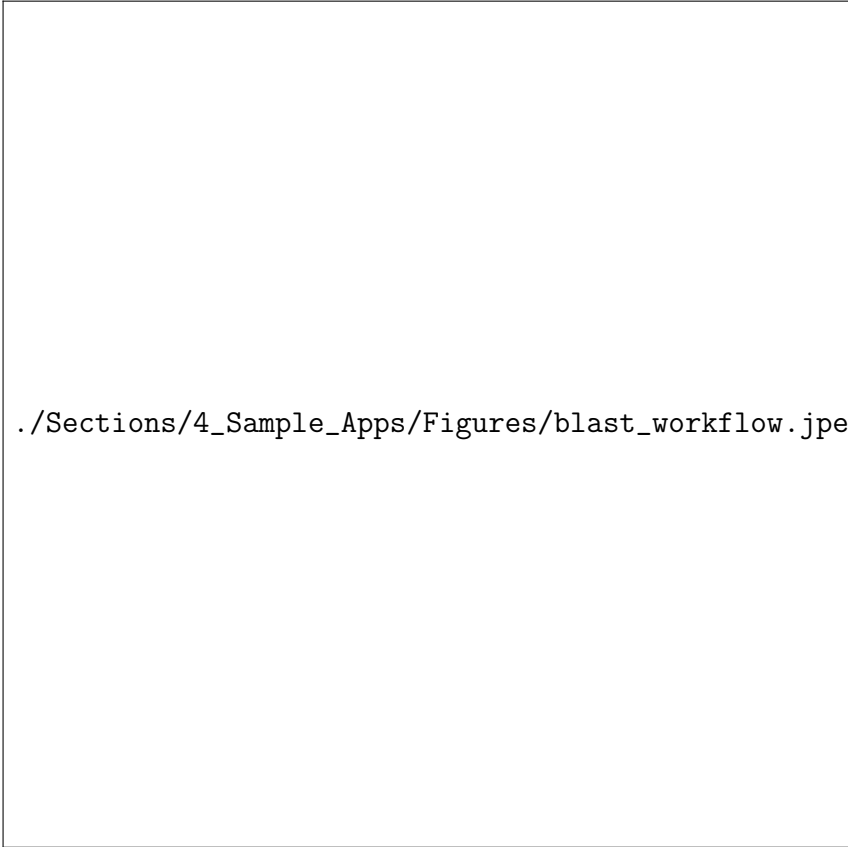
```
user@bsccompss:~$ cp ~/workspace/sparselu/package/SparseLU.tar.gz /home/  
user/  
user@bsccompss:~$ tar xzf SparseLU.tar.gz  
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/sparselu.jar
```

The command line to execute the application:

```
user@bsccompss:~$ runcompss sparselu.SparseLU <matrix_dim>
```

4.3 BLAST Workflow

BLAST is a widely-used bioinformatics tool for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences with sequence databases, identifying sequences that resemble the query sequence above a certain threshold. The work performed by the COMPSs Blast workflow is computationally intensive and embarrassingly parallel.



./Sections/4_Sample_Apps/Figures/blast_workflow.jpeg

Figure 3: The COMPSs Blast workflow.

The workflow describes the three blocks of the workflow implemented in the **Split**, **Align** and **Assembly** methods. The second one is the only method that is chosen to be executed remotely, so it is the unique method defined in the interface file. The **Split** method chops the query sequences file in N fragments, **Align** compares each sequence fragment against the database by means of the Blast binary, and **Assembly** combines all intermediate files into a single result file.

This application uses a database that will be on the shared disk space avoiding transferring the entire database (which can be large) between the virtual machines.

```
user@bsccompss:~$ cp ~/workspace/blast/package/Blast.tar.gz /home/user
/
user@bsccompss:~$ tar xzf Blast.tar.gz
user@bsccompss:~$ export CLASSPATH=$CLASSPATH:/home/user/blast.jar
```

The command line to execute the workflow:

```
user@bsccompss:~$ runcompss blast.Blast <debug> <bin_location>
                                <database_file>
                                <sequences_file>
                                <frag_number> <tmpdir>
                                <output_file>
```

Where:

- **debug**: The debug flag of the application (true or false).
- **bin_location**: Path of the Blast binary.
- **database_file**: Path of database file; the shared disk **/sharedDisk/** is suggested to avoid big data transfers.
- **sequences_file**: Path of sequences file.
- **frag_number**: Number of fragments of the original sequence file, this number determines the number of parallel Align tasks.
- **tmpdir**: Temporary directory (**/home/user/tmp/**).
- **output_file**: Path of the result file.

Example:

```
user@bsccompss:~$ runcompss blast.Blast true
/home/user/workspace/blast/binary/blastall
/sharedDisk/Blast/databases/swissprot/swissprot
/sharedDisk/Blast/sequences/sargasso_test.fasta 4 /tmp/
/home/user/out.txt
```

Please find more details on the COMPSs framework at
`http://compss.bsc.es`