CFX-5 Solver Parallelisation

Introduction to CFX-5 Parallel

The parallel implementation of the CFX-5 Solver is based on the *Single-Program-Multiple-Data* (SPMD) model. This model runs identical versions of the code on one or more processors.

The overall parallel run procedure is divided into two steps:

- a *partitioning* step, where the mesh is divided into a number of different segments, or partitions, and,
- a *running* step, where the mesh partitions are solved using separate processes (a *master* and one or more *slave* processes), often on different machines, with each process working on its own partition.

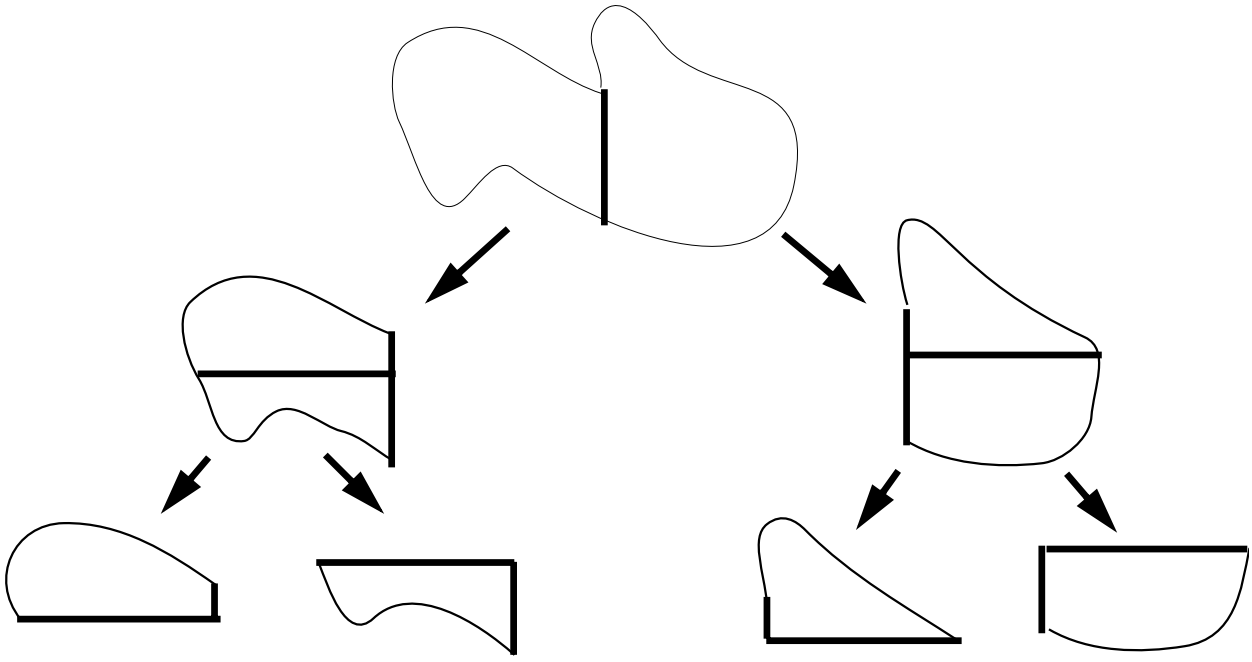
The CFX-5 Solver is designed so that all of the numerically intensive tasks are performed in parallel, and administrative tasks such as simulation control and user interaction, as well as the input/output phases of a parallel run, are performed in sequential mode by the master process. This approach guarantees good parallel performance and scalability of the parallel code, as well as ensuring that the input/output files are like those of a sequential run.

The communication between processes during a parallel run is performed using the PVM (Parallel Virtual Machine) or MPI message-passing libraries. In the PVM concept, the processes of a parallel run are distributed among the processors in the specified pool of hosts. PVM internally takes care of the different hardware architectures, single and multi-processor environments and shared memory concepts.

This section describes the ideas behind parallel processing in CFX-5 and some advice on using it effectively. More detailed instruction on how to set up a parallel run can be found in **Setting Up and Running a Parallel Run (p. 52 in CFX-5 Solver and Solver Manager)**, and a tutorial example is available: **Setting Up to Run in Parallel (p. 217 in CFX-5 Tutorials)**.

Partitioning

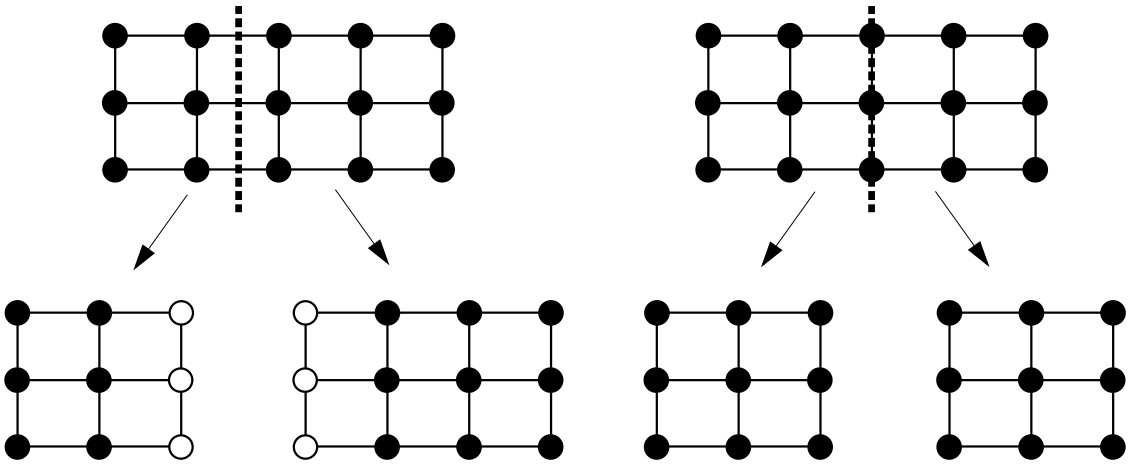
Partitioning is the process of dividing the mesh into a number of ‘partitions’ each of which may be solved on a separate processor. Several partitioning methods have been developed, but most are based on recursive bisection and differ only in the bisection step. The original mesh is first decomposed into two meshes of approximately equal size. The decomposition is then repeated recursively until the required number of partitions is obtained. The diagram below shows this process.



See **The CFX-5 Partition File (p. 126)** for details on viewing the mesh partitions after running the CFX-5 Solver in Partition Only mode.

Element Based and Node Based Partitioning

Any three-dimensional mesh can be partitioned using either element- or node-based partitioning. Element-based partitioning divides the mesh along element faces without dividing elements themselves, i.e. at nodal locations. Node-based partitioning divides the mesh across element faces. i.e. between nodal locations.



Node based partition

Element based partition

CFX-5 uses node-based partitioning because this is consistent with the node-based linear solver. There are three available partitioners:

- MeTiS
- Recursive Coordinate Bisection
- User Defined Direction

Multilevel Graph Partitioning Software - MeTiS

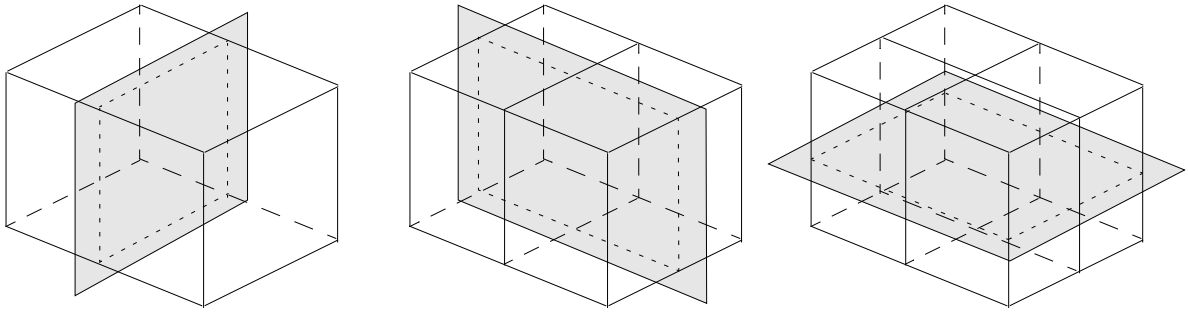
The public domain partitioning package MeTiS (Karypis and Kumar, 1996) uses the currently most advanced mesh partitioning algorithm, the Multilevel Graph Partitioning Algorithm. The basic idea behind this algorithm is as follows: a graph is built containing the topology information of the mesh to be partitioned. This graph is first coarsened down to a few hundred vertices. A bisection of the resulting much coarser graph is calculated and then the resulting partitions are projected back onto the original graph, by consecutively refining the partitions.

The MeTiS partitioner uses a fast algorithm which creates very efficient partitions. The whole process is fully automatic. However, it is unable to take advantage of coordinate direction alignment, and a substantial amount of dynamically allocated memory is required to run it (approximately 250 MB for a mesh containing 1.0E+06 nodes).

MeTiS is the default partitioner used by CFX-5.

Recursive Coordinate Bisection

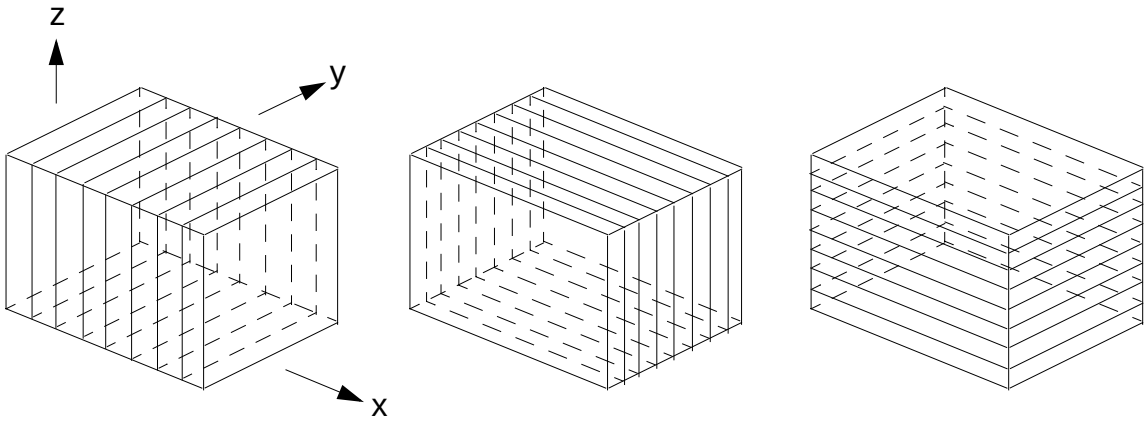
The Recursive Coordinate Bisection partitioning algorithm is very efficient with respect to CPU time and additional memory. The partitioning is based on the global coordinates of the mesh. Each step of the recursive bisection is performed in the coordinate direction with the largest dimension.



Like MeTiS, the Recursive Coordinate Bisection partitioner uses a fast, fully automatic algorithm, but it has only a small additional memory overhead. A particular disadvantage is that larger overlap regions can exist compared with the MeTiS partitioner, and each partition may contain separate parts.

User Defined Direction

The simplest partitioning algorithm is coordinate based in a direction whose vector components can be specified directly. This partitioning method uses the most efficient algorithm, and also has small additional memory overhead.



The partitioning process itself is partially controllable through the direction specification. However, in general, larger overlap regions occur compared with both the MeTiS and recursive coordinate bisection partitioners.

Required Hosts files and user setup

Full instructions on setting up parallel features in CFX-5 are given in **Parallel Setup (p. 71 in Installing & Introduction to CFX-5)** (UNIX) and **Parallel Setup (p. 89 in Installing & Introduction to CFX-5)** (Windows).

System Hosts File

Running CFX-5 in parallel mode requires a file which contains information about all available hosts which can be used in a parallel run. As part of a standard CFX-5 installation procedure, a file, called `hosts.ccl`, is created as the *System Hosts File*. This has to be created by your system administrator, using an appropriate editor. See **hosts.ccl file (p. 74 in Installing & Introduction to CFX-5)** (UNIX) or **hosts.ccl File (p. 91 in Installing & Introduction to CFX-5)** (Windows) for more details.

The hosts used for a parallel run in CFX-5 are specified in the EXECUTION CONTROL section of the Command Language for the run, in the same format as the system hosts.ccl file.

User setup for parallel runs

In order to run in parallel, hosts also have to be able to communicate with each other without requiring password authentication for different machines, using the `rsh` (remote shell) protocol.

UNIX

You can check whether you can do this on a UNIX system by executing the following command on the machine which you want to run as the master:

```
rsh <machine_name> echo OK
```

(**remsh** on HP systems) where <*machine_name*> is the name of any UNIX machine you want to run a process on during your parallel run, including the master machine itself. If this fails with message "`Permission denied`", you must add a line of the form,

```
<master> <userid>
```

to the `.rhosts` file on the slave machine. Creating an `.rhosts` file is described in **Remote Access (p. 143 in Installing & Introduction to CFX-5)**.

Windows

You can check whether you can do this on a Windows system by executing the following command at a command prompt on the machine which you want to run as the master:

```
rsh <machine_name> cmd /c echo OK
```

where <`machine_name`> is the name of any Windows machine you want to run a process on during your parallel run, including the master machine itself. If this fails with message "`Connection refused`", ensure that a `rsh` service is running on the machine called <`machine_name`>. How to set up an `rsh` service is described in **Parallel Setup (p. 89 in Installing & Introduction to CFX-5)**.

Message Passing Interface (MPI) for Parallel

When to Use MPI:

- Do not use MPI mode for network parallel simulations. PVM mode (the default parallel communication mode) is far superior in terms of robustness to abnormal situations for network parallel runs (e.g. a computer crash during a run).
- Only consider using MPI on multi-processor (MP) machines (computers which contain two or more internal CPU's). In this case determine which of PVM or MPI modes runs faster for your typical application. Performance is computer and application specific.
- Check that the operating system kernel has a large enough number of "semaphores" available. The sections that follow explain what semaphores are, and what you need to know about them in order to run MPI successfully on your system.

How to Use MPI:

By default all parallel mode simulations run in PVM mode. To change to MPI mode you must do one of the following:

- Start the solver monitor from the command line, with the "**`-parallel-mode mpi`**" argument. For example, from the UNIX command line:

`cfx5solve -parallel-mode mpi`

This will affect the runs that are subsequently started with the solver manager or from the solver script directly.

- Modify the following line in the solver manager preferences file, stored off of your home directory in the file `~/.cfx/cfxsolman/userPreferences`:

`ShowParallelModeSelector = 1`

This switch is by default set to 0, which removes the parallel mode selector from the "Define Run" panel of the solver monitor. When **`ShowParallelModeSelector = 1`** is set, and the Run Mode is set to Local Parallel or Distributed Parallel, then a Parallel Mode panel appear and permits you to select either PVM or MPI run mode.

Semaphores and shared memory segments:

- Semaphores are constructs used by MPI to enable interprocess communication.
- Semaphores are the shared-memory equivalent to "sockets" that are used by PVM to enable interprocess communication.
- Each parallel process uses its own semaphore.
- Shared memory access via semaphores is managed by the operating system kernel.
- Semaphores are created and used by a user-id, but managed by the operating system.
- When a CFX-5 MPI parallel job is finished, these semaphores are, by default, deleted. If the job terminated in some abnormal fashion, for example one of the processes was killed, then the semaphores in use are not deleted and remain in an idle state. This situation consumes free semaphores on your system.
- There is a maximum limit on the number of semaphores that the operating system will permit. Once this limit is reached, no new semaphores can be created.
- CFX-5 behaves erratically in various ways when the semaphore limit is reached.

Typical problems when you run out of semaphores:

- You cannot start a new MPI run. Typical error message to console:

```
p0_97244:  p4_error: semget failed for setnum=%d: 0
      An error has occurred in cfx5solve:
```

- A new MPI job starts up, but kills an already running MPI job.
- When two MPI jobs are running at the same time, for the same user-id, stopping one MPI job kills the second MPI job.

Checking how many semaphores are in use, and by whom:

There is a built-in command for UNIX: "**ipcs**", or "**ipcs -a**". Use this command to see what semaphores are owned by your user -id. Typical output is as follows:

```
machine.company.com: ipcs

Message Queues:
T       ID        KEY    MODE         OWNER    GROUP
q        0 0x416d02d8 --rw-------      root   system

Shared Memory:
T       ID        KEY    MODE         OWNER    GROUP
m      768    0x7a08e --rw-------       joe    all
m     1409    0x57463 --rw-------       moe    all

Semaphores:
T       ID        KEY    MODE         OWNER    GROUP
s        0 0x416d02d8 --ra-------      root   system
s        1 0x416d029e --ra-------      root   system
s       82    0x7a08e --ra-------       joe    all
s       83    0x7a08f --ra-------       joe    all
s       84    0x7a090 --ra-------       joe    all
s       85    0x7a091 --ra-------       joe    all
s      214    0x57463 --ra-------       moe    all
s      183    0x57464 --ra-------       moe    all
s      184    0x57465 --ra-------       moe    all
s      185    0x57466 --ra-------       moe    all
```

The user `joe` has one shared memory segment and 4 semaphores left over from an abnormally aborted previous CFX-5 computation. These need to be deleted.

Deleting the semaphores you are using:

Once the system limit is hit for semaphores, new MPI jobs are not possible. Since MPI does not delete these semaphores if a simulation has a abnormal termination, it is up to each user to manually clean up (delete) all such semaphores. It is also a good idea to also delete any remaining shared memory segments owned by your account at the end of an aborted run. This leaves the system free for other users to use

for their MPI runs. Continuing from the above example, the user "joe" would issue the UNIX command "**ipcrm -m xxx -s yyy**" where **xxx** is shared memory ID number and **yyy** is a semaphore ID number. For example,

```
machine.company.com: ipcrm -s 82 -s 83 -s 84 -s 85 -
m 768
```

This will delete the semaphores and the shared memory segment, leave the system free for the next user. On some systems the exact syntax of the **ipcrm** command may differ from that shown above.

Alternatively, you can use a supplied script that finds all semaphores and shared memory segments owned by you, and deletes them automatically. The script is found in the **Tools** directory of your installation, and is executed as follows:

```
machine.company.com: $CFX_ROOT/tools/share/bin/
cleanipcs
```

Be aware that if you run this command while you have a current MPI job running, that this job will terminate abnormally. You can only run this command when you know that you are not running any MPI jobs at that time.

Checking the maximum number for your system:

It may be necessary to increase the maximum number of semaphores on your system, so that several users/jobs can each run in parallel, simultaneously. Once the total number of semaphores are used, then more parallel MPI jobs can be started. You can find out the maximum number of semaphores on your system with the following commands:

1. Linux: **sysctl -a | grep -i sem**

2. Sun: **sysdef | grep -i semaphore**

3. DEC: **/sbin/sysconfig -q ipc**

4. SGI: **/usr/sbin/systune sem**

5. HP: **/user/sbin/sysdef | grep -I sem** (may fail on certain systems)

6: IBM: not known at this time.

Increasing the maximum number of semaphores for your system:

In general this is not a simple task, and it varies greatly between different computer vendors and operating systems. It may involve changing a system resource file and rebooting the computer, or it may involve making changes to the system kernel and recompilation of the system kernel. Contact your system administrator if you need to increase the number of semaphores on your system.

Advice on using CFX-5 in parallel

Optimising mesh partitioning

Partitioning in CFX-5 Parallel is a pre-processing step. Using the Solver Manager, you can mesh partitioning according to:

- the partitioning method
- the number of partitions

You can select from one of three different partitioning methods, and create up to a maximum of 128 partitions for your simulation.

In general, you should try to follow these guidelines wherever possible:

Don’t run small jobs in parallel

You are unlikely to see any performance increases in solution time if your mesh contains less than 10,000 nodes. Models with meshes smaller than this should be run in serial mode.

Always try to use the MeTiS partitioning method

MeTiS generates the best partitions with respect to the size of the overlap regions between the partitions, and is therefore the favoured, and the default, partitioning method in CFX-5. Use one of the alternative partitioning methods only if MeTiS fails, or if the memory overheads are unacceptably high.

Meshes containing Periodic Pair boundary conditions in particular are handled much more efficiently by MeTiS than the other partitioners.

Try to use a sensible number of partitions

The partitioning of a mesh leads to the creation of overlap regions at the partition interfaces. These regions are responsible for communication and memory overhead during a parallel run. During partitioning, CFX-5 prints partitioning diagnostic information to the

Solver Manager text window and Output File about partition overlaps. The percentage of overlap nodes to the total number of mesh nodes should ideally be less than 10% for efficient partitioning. Values greater than 20% will impair performance and are not recommended.

If necessary, store the partition file

As mentioned above, partitioning is a pre-processing step. By default, CFX-5 performs mesh partitioning automatically in combination with a parallel run. The creation of a partition file is therefore treated as an intermediate step, and is deleted at the end of a parallel run.

Under certain circumstances it is advisable to store the partition file permanently, by performing a partition-only run of the CFX-5 Solver. These are:

- when the machine on which the master process is to be run does not have enough memory to run the partitioner.
- when several parallel runs using the same number of partitions are to be performed.

Optimising the Parallel Run

The following is a bulleted list of tips to help you get the most from a parallel run using the CFX-5 Solver:

- For a single parallel run, try to use machines which have a similar level of performance.
- Avoid using swap space. Try to use machines which can perform their assigned tasks using their own physical memory.
- If you are using a network of workstations, use high speed ethernet connections (100 Mbit is preferred to 10 Mbit).
- Before starting a parallel job, check to see if your selected machines or processors can be used exclusively. If you have to share their use with others this may increase the cpu static load.
- Using two processors on a multi-processor machine is generally faster than using two processors on different machines.
- Try to avoid assigning more than one process per processor.

- Use a sensible number of partitions. There should always be a balance between the individual partition size and the number of partitions.

Error Handling

This section gives suggestions on what to do if you encounter problems during a parallel run of the CFX-5 Solver. Most failures occur during the initial phase of the run as a result of configuration or PVM problems.

Problems with PVM

- Check to see whether PVM daemon files already exist on the selected hosts. On UNIX systems, these files are usually located in the `/tmp` directory and have the name `pvmd.`*`<user_id_number>`*, where *`<user_id_number>`* is your own user id number. On Windows systems, these files are usually located in `c:\temp` and have the name `pvmd.`*`<username>`*, where *`<username>`* is your own user name.

- Avoid using machines that share the same `/tmp` directory in the parallel environment.

- Try to start the PVM console on the master machine and then **add** the desired hosts. On a UNIX system, you can start the PVM console by typing the command `pvm` in a UNIX terminal window - you may need to set the environment variable `PVM_ROOT` to ***`<CFXROOT>`***`/tools/pvm` before being able to run the PVM commands. On a Windows system, you need to run ***`<CFXROOT>`***`/tools/CYGWIN32_NT/pvm3.4/console/win32/pvm.exe` to start the PVM console with an argument that is the full path to your PVMhosts file. On a typical installation, this is ***`<CFXROOT>`***`/config/`***`<release>`***`/PVMhosts.`

  The following are some of the main PVM commands:

  `add` - adds a specified host or list of hosts.

  `conf` - shows the current host configuration.

  `quit` - leave the PVM environment and keep PVM running in the background.

halt - leave and stop the PVM environment.

- Check if the PVM architectures of the selected hosts are supported by your installation of CFX-5. On a UNIX system, the PVM architecture can be checked by running

  ***<CFXROOT>*/tools/pvm/lib/pvmgetarch**

  on each machine. On a Windows machine, the PVM architecture is always WIN32.

- Check that you have a valid account on each of the selected machines. On UNIX systems, each of these accounts must have the same user name.

- If you are working on a Windows system you may occasionally get the error message like the one below if you run two copies of the PVM daemon one after the other within a short time, or after using the `rsh` command:

```
hoster() 1 to start, wait id 262168
0. t80000 machinename so=""
Bound to port 1023
Bound to port 1022
Cannot connect to RSH port: 10048
error sending command: number 10057
error in receive on socket: The socket is not connected.

could not shut down rsh client socket
```

  To overcome this problem, you should wait for two or three minutes and try again.

- The PVMHosts file is only read when the PVM daemon is first started. This means that if you change the PVMHosts file while any parallel run is running, no changes will be noticed until the next time that all parallel processes have stopped running.

- on Windows, you may need to kill the pvmd3.exe process using the Task Manager

- on UNIX, you may need to kill the pvmd process.

Problems with the CFX-5 Executables

- Check the Hosts File (`hosts.ccl`) for syntax errors.
- Check that the specified executables for each host in the Hosts File exist, and that the files have execute permission set.
- Check that the specified executables are correct for the specified hosts (operating system, 32 bit or 64 bit executables, etc.).
- If a slave process is running on a Windows system, and the parallel run complains that it cannot find the solver executable, even though the specified executable exists, you should check to see that the user who is running the `rshd.exe` process (the RSH Service) has appropriate permission to run the specified solver executable.

  You can find out which user is running the RSH Service as follows. Log on as `administrator`. Select **Settings>Control Panel** from the Start menu, and double-click on the **Services** icon. In the **Services** panel, double-click on the **Remote Shell (RSH) Service** line. A **Service** window will pop up. The user running the RSH Service has the username which appears as the **Log On As** setting.

Problems with CFX-5 Licenses

- Check that you have the correct parallel licenses to run a parallel job on the assigned hosts.
- Check that you have enough licenses for the specified number of partitions.

rsh Problems

- If you are running on a Windows workstation, check to make sure that you are using the right version of `rsh`. This is described in **Setting up the rsh Service (p. 94 in Installing & Introduction to CFX-5)**.

Convergence Problems

Occasionally you may find that jobs submitted in serial will converge while those in parallel fail. This can be due to the different internal structure of the multigrid solver. The partitioned mesh leads to different coarse mesh blocking than the serial mesh, and if you have selected a timestep size which is close to the critical convergence limit this can cause convergence problems.

Usually a reduction in the timestep size alleviates this problem.

Measuring Parallel Performance

In general, there are two reasons why you might wish to run a CFX-5 job in parallel:

- Getting results faster by combining the processing power of two or more processors. The performance of this set up is measured by the system *speedup*.
- Performing simulations which require more memory than that which can be provided by a single machine. The performance of this feature is best measured using the *memory efficiency*.

In a real-world computer environment, it is not always easy to measure parallel performance. Measured execution times depend on relative speed of processors, which in turn is limited by the processor architecture as well as its load. For heterogeneous networks, the parallel performance is directly determined by the speed of the slowest processor.

Wall Clock Performance

In order to measure wall clock performance, two parameters can be defined:

$$Speedup, S = \frac{T_s}{T_p}$$

$$Efficiency, E = \frac{T_s}{NT_p} = \frac{S}{N}$$

where the subscripts s and p refer to the sequential and parallel wall clock execution times respectively, and N is the number of partitions. The best wall clock performance increase which can be expected is a linear speed up ($S = N$), and this corresponds to an efficiency of 100%.

The performance determination can sometimes not be straightforward due to the fact that only part of the CFX-5 Solver run is actually performed in parallel. The reading and distributing of the Definition File data, and the collecting and writing of Results File data are highly I/O dependent and not parallelised. These stages therefore depend on high disk speeds and fast network communication for fast operation, and should be neglected in any parallel performance evaluation. The time taken for the calculation stage of the CFX-5 Solver can be estimated from the difference between the times on the following lines of the Output File:

```
CFD Solver started: Wed Oct 28 12:10:39 1998
.......
CFD Solver finished: Wed Oct 28 16:37:08 1998
```

Memory Efficiency

The memory efficiency,

$$E_m = \frac{M_s}{M_p}$$

where,

$$M_p = \sum_{i=1}^{N} M_{p,i}$$

can be used to compare the total memory required by all processes of a parallel run, M_p, with the memory required by that of the equivalent serial run, M_s. The memory efficiency for CFX-5 will always be smaller than 100%, but should generally be in the range of 80-95% for meshes with reasonable partitions. With respect to workstation clusters, it is important to know that all processes (operating on both master and slave machines) use a nearly identical amount of memory.

Detailed information about the memory requirements of a parallel run are written to the Output File.

Visualising Mesh Partitions

Partitioning information is appended to the Results File during a parallel run of the CFX-5 Solver. You can view the partitions in CFX-Post by loading the Results File and viewing the variable **Real partition number** on a locator to display the partitions of the mesh.

If you want to view the mesh partitions without performing a parallel run, you can concatenate the Partition File to the Definition File using

```
cat file.def file.par > new_file
```

on UNIX systems, or

```
copy /b file.def + file.par new_file
```

at a Windows command prompt, and read the information in the new file into CFX-Post.