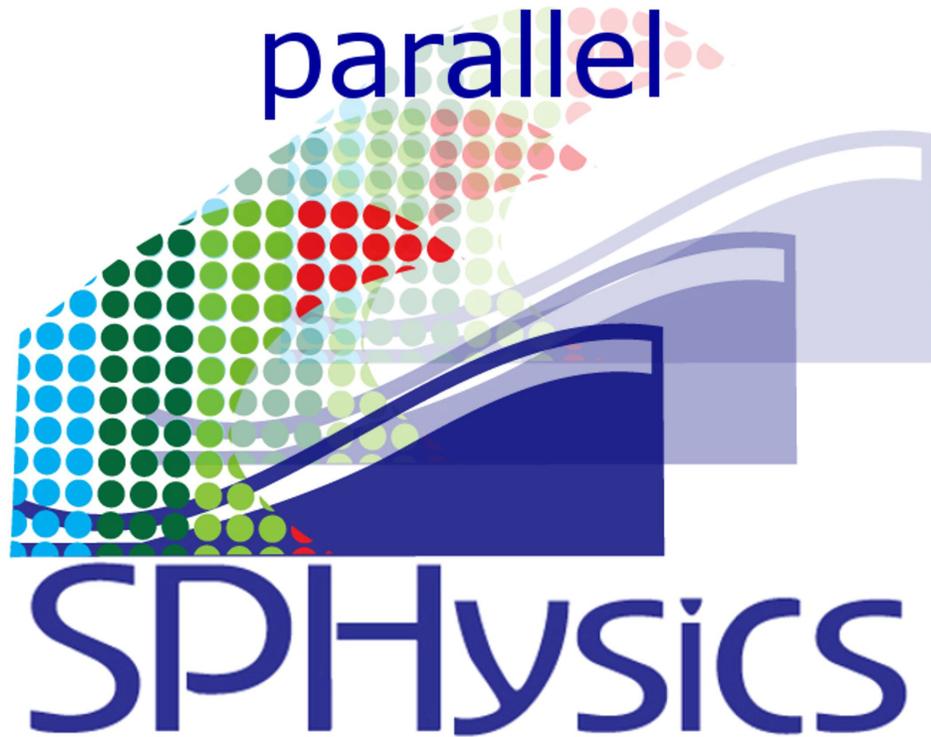


# User Guide for parallelSPHysics v2.0 using MPI



January 2011

## Abstract

The code parallelSPHysics has identical functionality as the serial SPHysics code but has been design to perform simulations of millions of particles. The code is parallelised using the MPI formalism, and thus requires MPICH or OpenMPI to be installed on your parallel machine. Full information on the SPHysics code can be found in the guide for the serial code. This document describes (i) how to run the parallel code on unix/linux-based operating systems, (ii) the main changes from the serial code, (iii) the test cases which are the same as the serial code but with more particles, and (iv) the performance which shows superlinear speedup up to 64 cores in 2-D and 48 cores in 3-D.

B.D. Rogers ([benedict.rogers@manchester.ac.uk](mailto:benedict.rogers@manchester.ac.uk))

R.A. Dalrymple ([rad@jhu.edu](mailto:rad@jhu.edu))

M. Gómez Gesteira ([mggesteira@uvigo.es](mailto:mggesteira@uvigo.es))

A.J.C. Crespo ([alexbexe@uvigo.es](mailto:alexbexe@uvigo.es))



## **Acknowledgements**

The development and application of SPHysics were partially supported by:

- Xunta de Galicia under project PGIDIT06PXIB383285PR.
- Office of Naval Research, Geosciences Program
- EPSRC Project Grant GR/S28310
- ESPHI (An European Smooth Particle Hydrodynamics Initiative) project supported by the Commission of the European Communities (Marie Curie Actions, contract number MTKI-CT-2006-042350).
- Flood Risk Management Research Consortium (FRMRC) Phase 2, EPSRC Grant F020511
- Research Councils UK (RCUK) Research Fellowship

# CONTENTS

<b>1. INTRODUCTION .....</b>	<b>5</b>
1.1 Smoothed Particle Hydrodynamics and the need for hardware acceleration.....	5
1.2 The Structure of this Guide.....	5
<b>2. HOW TO RUN THE CODE.....</b>	<b>7</b>
2.1 Introduction.....	7
2.2. Installation.....	7
2.3. Program Outline.....	8
2.3.1. SPHysicsgen .....	8
2.4 Generating the geometry.....	9
2.5 Compiling and Running the executable.....	10
2.5.1 Submission script files (SGE & LSF systems) .....	11
2.5.2 Submitting consecutive jobs: Job dependencies and Checkpointing.....	12
2.6 Extra Options in the CaseN.txt files for parallel code .....	13
<b>3. CHANGES FROM SERIAL CODE.....</b>	<b>14</b>
3.1 Differences between the serial and parallel codes .....	14
3.2 Grid sweep .....	14
3.3 Differences in Naming Conventions.....	19
3.4 Storage of Particle Data on each Processor: varying particle numbers .....	19
3.5 Pointers .....	20
3.6 Boundary Particles .....	20
3.7 New Subroutines .....	21
3.8 Dynamic Load Balancing & MPI Topologies (1-D, 2-D & 3-D).....	22
3.9 Choosing the Number of Processors: issues and advice .....	24
<b>4. TEST CASES .....</b>	<b>25</b>
4.1 Introduction.....	25
4.2 Test Case resolutions, particle numbers and Default processor numbers.....	25
4.2.1 2-D Simulations .....	25
4.2.2 3-D Simulations .....	26
<b>5. PERFORMANCE.....</b>	<b>27</b>
5.1 Performance of 2-D parallelSPHysics .....	27
5.2 Performance of 3-D parallelSPHysics .....	29
<b>6. VISUALISATION .....</b>	<b>31</b>
<b>7. COMPLEX GEOMETRIES.....</b>	<b>31</b>
7.1 Format of input files and Example Setup .....	31
<b>8. FUTURE DEVELOPMENTS .....</b>	<b>32</b>
<b>9. REFERENCES.....</b>	<b>32</b>

# 1. INTRODUCTION

## 1.1 Smoothed Particle Hydrodynamics and the need for hardware acceleration

Smoothed Particle Hydrodynamics (SPH) is revolutionising what can be simulated in both fluid dynamics and solid mechanics due to its meshless nature. The method has opened up the possibility of modelling phenomena that may exhibit highly nonlinear behaviour existing over a range of time and length scales (see the 2010 Special Issue of *Journal of Hydraulic Research* for a good overview of current activity in this field).

Despite the successes, as a methodology SPH suffers from numerous drawbacks. Perhaps one of the most inhibiting has been the excessive runtimes. In 2-D each particle will interact with maybe 50 particles that reside within its kernel, while in 3-D, this number can reach nearly 200 depending on the size of the smoothing length, use of adaptive kernels, etc. Due to the restrictions to maintain accuracy within SPH, such as partition of unity, simulations of real problems (i.e. anything other than toy problems) need a very large number of particles in the region of many millions ( $10^6 - 10^{10}$  particles). For a code that runs on a single processor (i.e. 1 CPU), this is completely unfeasible both in terms of memory required and the simulation runtimes. Hence, the requirement for hardware-accelerated SPH codes has become an integral part of the development of SPH as useful engineering simulation tool.

Numerous codes for running on parallel machines now exist. In this guide we present the user instructions for the parallel version of SPHysics which has been developed for simulating free-surface flows, specifically those encountered in coastal and shallow water hydrodynamics. The nature of HPC is that the best performance can only be achieved with a code optimised to individual architectures which was not the objective here, i.e. a portable parallel SPH code.

Note, parallelSPHysics has been tested up to  $10^7$  particles and will not be able to simulate  $10^{10}$  particles.

## 1.2 The Structure of this Guide

As the functionality of the parallelSPHysics is the same as the serial code, all descriptions of the theoretical basis are described in the main SPHysics user guide (Gomez-Gesteira et al. 2010). This guide is therefore structured as follows. First, in Chapter 2, we will describe how to run the code on a unix/linux-based operating system, and then in Chapter 3 we describe the main differences between the serial code and the new parallel code. In Chapter 4, we briefly run through the test cases which are identical to the cases that come with the serial code, but using a finer resolution, i.e. a larger number of particles. In Chapter 5, we describe the performance of the parallel code in terms of speed up and efficiency. In Chapters 6, 7 & 8 we describe visualisation (same as serial code), generating complex geometries in 2-D (new) and future developments, respectively.



## 2. HOW TO RUN THE CODE

### 2.1 Introduction

Unlike the serial code, launching the parallel code will depend heavily on the architecture you using. As mentioned in the abstract you will need either MPICH or OpenMPI installed in order to run the MPI formalism (see <http://www.mcs.anl.gov/research/projects/mpich2/> or <http://www.open-mpi.org/> ). The code has been developed with the assumption that most parallel environments are not using Windows, but are running a linux/unix operating system (see Top 500 HPC Machines: <http://news.bbc.co.uk/1/hi/technology/10187248.stm>).

For the installation of the code, some information is repeated here from the serial version of the code.

### 2.2. Installation

Two versions of SPHysics are available in this release:

- **parallelSPHysics\_2D**. The computational domain is considered to be 2D, where  $x$  corresponds to the horizontal direction and  $z$  to the vertical direction.
- **parallelSPHysics\_3D**. The computational domain is fully 3D,  $x$  and  $y$  are the horizontal directions and  $z$  the vertical direction.

SPHysics is distributed in a compressed archive (.zip). Decompress each archive in the desired location.

The directory tree shown in Figure 3.1 in the serial SPHysics guide (Gomez-Gesteira et al. 2010) can be observed after uncompressing the distributed compressed files (.zip).

In that figure, the following directories can be observed both in 2D and in 3D.

**source** contains the FORTRAN codes. This directory contains two subdirectories:

SPHysicsgen: contains the FORTRAN codes to create the initial conditions of the run.

SPHysics: contains the FORTRAN source codes of SPH.

**execs** contains all executable codes.

**run\_directory** is the directory created to run the model. The different subdirectories *Case1*, ..., *CaseN* placed in this directory correspond to the different working cases to be created by the user. Input and output files are written in these directories

**Post-Processing** this directory contains codes to visualize results (copy files from here to the working *Case* directory for visualisation).

### 2.3. Program Outline

Both the 2D and 3D version consist of two programs, which are run separately and in the following order.

2D Code:

SPHYSICSgen\_2D: Creates the initial conditions and files for a given case.

SPHYSICS\_2D: Runs the selected case with the initial conditions created by SPHYSICSgen\_2D code.

3D Code:

SPHYSICSgen\_3D: Creates the initial conditions and files for a given case.

SPHYSICS\_3D: Runs the selected case with the initial conditions created by SPHYSICSgen\_3D code.

In general, 2D or 3D appended to the source file name means that the source is suited for 2D or 3D calculations.

In the remainder of this document, SPHYSICSgen and SPHysics, when used, refer to both the aforementioned 2D and 3D programs for convenience. For example, SPHYSICSgen will refer to both SPHYSICSgen\_2D and SPHYSICSgen\_3D.

#### 2.3.1. SPHYSICSgen

All subroutines are included in two source files (SPHYSICSgen\_2D.f or SPHYSICSgen\_3D.f), depending on the nature two or three- dimensional of the calculation. Each source uses a different common file, where most of the variables are stored. The common files are common.gen2D (in 2D) and common.gen3D (in 3D). Both versions (2D and 3D) can be compiled by the user with any FORTRAN compiler and the resulting executable file is placed in subdirectory *\execs*.

SPHYSICSgen plays a dual role: (i) Creating the MAKEFILE to compile SPHysics; and (ii) Creating the output files that will be the input files to be read by SPHysics. These files contain information about the geometry of the domain, the distribution of particles and the different running options.

For example, *SPHYSICSgen* can be executed using one of the following two commands:

##### 1. *SPHYSICSgen < input\_file*

*input\_file* is the general name (any name can be used) of the file containing the running options. Different examples of *input\_file* will be shown in next section.

## 2. *SPHYSICSgen*

In this case, data about the run must then be provided by the user by means of the keyboard and the information about the run appears on the screen. This option can be used by beginners to get familiarized with the different options.

### 2.3.1.1. Creating compiling options

The compilation of *SPHysics* code depends on the nature of the problem under consideration and on the particular features of the run. Thus, the user can choose the options that are better suited to any particular problem and only those options will be included in the executable versions of *SPHysics*. This protocol speeds up calculations since the model is not forced to make time consuming logical decisions (i.e. if statements) – see Section 3.2 of the main guide (Gomez-Gesteira 2010). There are only two new options to choose the MPI FORTRAN compiler and whether to activate optimization of the MPI partitioning (i.e. load balancing) which are described in Section 2.6.

### 2.3.1.2 Output files from *SPHYSICSgen* identical to serial code

The same identical output files are generated in *SPHYSICSgen* as for the serial code version with a few extra files for the parallel code. The identical files are: [SPHYSICS.mak](#), [INDAT](#), [IPART](#), [matlabin](#), [NORMALS](#), [OBSTACLE](#), [WAVEMAKER](#), [GATE](#), [Tsunami\\_Landslide.txt](#), [Floating\\_Bodies.txt](#). The reader is referred to the serial guide for the content of these files

### 2.3.1.3 Output files from *SPHYSICSgen* solely for parallel code

The extra files are [MPI\\_container\\_Limits.txt](#) and [BoundaryPs\\_MPI\\_Pointer.init](#). As we mentioned above, different output files are created by *SPHYSICSgen*. These files can be used either by the *SPHysics executable* as input files or by MATLAB codes to visualize results (different MATLAB codes are provided in */Post-processing* subdirectory).

## 2.4 Generating the geometry

In a change to the serial code, the geometry is first generated separately from the launching of the parallel code.

Change to the `run_directory/CaseN/` of the case you would like to run.

At the command prompt, for gfortran enter:

```
./CaseNgen_unix_gfortran.bat
```

or for the Intel compiler, enter:

```
./CaseNgen_unix_ifort.bat
```

See also the complex 2-D geometry generator in Section 7.

## 2.5 Compiling and Running the executable

This part will depend heavily on your parallel architecture. The code has been developed on machines that use the LSF (Load Sharing Facility) and the SGE Submission systems. At present, the information to run parallel jobs on other environments is not known. However, you can use the “CaseN\_unix.bat” files as an example batch submission file which you can then alter to your system (see your system administrator).

In each “CaseN\_unix.bat” file, there is a choice of batch submission files or interactive job submission. You must comment/uncomment the batch file appropriately to select each one. With the code the script files: `script_bsub.bsub` and `script_qsub.qsub` have been provided for the LSF and SGE Submission systems, respectively. An example “Case3\_unix.bat” file is below which has the **default option of submitting using the SGE system**:

```
UDIRX=`pwd`
#
if [ $? -eq 0 ]; then
    SPHYSCSgen_Done="yes"
    echo 'SPHYSCSgen_Done = ' $SPHYSCSgen_Done
    rm SPHYSCS_2D
    cp SPHYSCS.mak ../../source/SPHYSCS2D/
    cd ../../source/SPHYSCS2D
    pwd
    ## - Uncomment following line for quicker repeated compilation -
    make -f SPHYSCS.mak clean
    make -f SPHYSCS.mak
    if [ $? -eq 0 ]; then
        echo ' '
        SPHYSCScompilationDone="yes"
        echo 'SPHYSCScompilationDone = ' $SPHYSCScompilationDone
        echo ' '
        rm SPHYSCS.mak
        cd $UDIRX
        #rm PART_*
        rm DETPART_*
        rm sph.out sph.error
        rm MPI_Partition_Positions.dat
        pwd
        cp ../../execs/SPHYSCS_2D ./

        ## Comment out the following lines as necessary depending on your architecture
        ## DON'T SELECT INTERACTIVE AT SAME TIME AS A BATCH SUBMISSION SYSTEM BELOW

        ## LSF Submission system
        ## - Interactive MPI execution -
        #prun -p login -n 4 -B 4 ./SPHYSCS_2D
        #
        ## - Batch Submission execution -
        #bsub < script_bsub.bsub

        ## SGE Submission system
        ## - Interactive MPI execution -
        #qsub -I -l nodes=4 ./SPHYSCS_2D
        ##
```

```

## - Batch Submission execution -
qsub script_qsub.qsub
else
  rm SPHYSICS.mak
  cd $UDIRX
  echo ' '
  echo 'SPHYSICS_2D compilation failed'
  echo 'Make sure correct compiler is selected in Case file'
fi
else
  cd $UDIRX
  echo ' '
  echo 'SPHYSICSgen failed'
fi

```



To compile and run the code using this file, at the command prompt enter:

```
./Case3_linux.bat
```

In this particular case, the SGE system will be used (invoked by the `qsub script_qsub.qsub` command).

Above is an example (rendered inactive by using the script comment sign #) for interactive submission enabled for LSF for 4 processors (`-n 4`) and will log into node 4 in order to launch (`-B 4`).

## 2.5.1 Submission script files (SGE & LSF systems)

### (i) SGE submission script (`script_qsub.qsub`)

Here is an example submission script for the SGE system which will submit a job to a queue called 'parallel' with 64 cores (processors) and will call the job SPHysics\_2D and write all output to a file called 'sph.out' with a file for error messages called 'sph.error':

```

#!/bin/bash

# -- the job is located in the current working directory :
#
#$ -cwd

# -- shell
#
#$ -S /bin/bash

# -- specify the queue
#
#$ -q mpich.q

# -- specify how many parallel processes I want :

```

```
#
#$ -orte.pe mpich 64

#$ -o sph.out
#$ -e sph.error

mpirun -np 64 ./SPHYSICS_2D
```

This script should be called `script_qsub.qsub` and submitted using the command as used above in the “Case3\_unix.bat” file above:

```
qsub script_qsub.qsub
```

### (ii) LSF submission script (`script_bsub.bsub`)

Here is an example submission script for the LSF system which will submit a job to a queue called ‘parallel’ with 32 cores (processors) with a maximum wall clock time of 23 hours 59 minutes and will called the job SPHysics\_2D and write all output to a file called ‘sph.out’ with a file for error messages called ‘sph.error’:

```
#BSUB -n 32
#BSUB -W 23:59
#BSUB -o sph.out
#BSUB -e sph.error
#BSUB -q parallel
#BSUB -J SPHysics_2D
#BSUB -B
prun -n 32 ./SPHYSICS_2D
```

This script should be called `script_bsub.bsub` and submitted using the command as used above in the “Case3\_unix.bat” file above:

```
bsub < script_bsub.bsub
```

## 2.5.2 Submitting consecutive jobs: Job dependencies and Checkpointing

Many parallel (supercomputing) clusters set a maximum wall-clock time limit for each job (e.g. 24 hours). In order to run a simulation that requires more than the wall-clock time limit, multiple submission scripts should be used with consecutive job dependencies, i.e. the submission of one script depends on the successful completion of a previous job. We do not describe this here, but we do point out that SPHysics does have the capability for checkpointing (i.e. multiple restarts) as an option in SPHYSICSgen for both the serial and parallel codes. For restarts the code uses new files generated during the running of the parallel SPHysics code: `NORMALS.RESTART`, `Floating_bodies.RESTART` and `BoundaryPs_MPI_Pointer.RESTART`.

## 2.6 Extra Options in the CaseN.txt files for parallel code

There are now three more options for running the code needed by the generation program `SPHYSICSgen_2D/3D.f` and the compiler chosen must be `mpif90`. These extra options are added only at the end of each `CaseN.txt` file.

Choice of compiler, `i_compile_opt` (5=`mpif90`), has been changed so that you can select `mpif90`. There is no other option in contrast to the serial code.

(i) Choice of MPI Cartesian Topology, `MPI_CartDims`.

For 2-D, there is the option of 1- and 2-dimensional topologies.

For 3-D, there is the option of 1-, 2- and 3-dimensional topologies.

(ii) Choice of automatic domain decomposition, `MPI_AutoDomainProcs` (1=automatic, 0=manual), AND if manual is chosen the number of processors in each topological direction must be specified, i.e. `n_procs_x0`, `n_procs_z0`

(iii) Choice of using adaptive MPI topology, `i_MPI_Adapt` (1=yes), with three parameters: `nFP_difference_max`, `n_itime_Adapt`, `n_ini_Adapt`:

`nFP_difference_max` = maximum difference in number of fluid particles between adjacent processors (default = 200)

`n_itime_Adapt` = number of timesteps when the position of the MPI partitions are checked to be moved, etc. (default = 500)

`n_ini_Adapt` = number of occasions when the position of the MPI partitions are checked to be moved, during initial optimisations before run starts (default = 5000)

These options are included in the extra input file "`MPI_container_Limits.txt`" whose contents are:

```
iCartTop_periodicity
xb_min,xb_max
yb_min,yb_max
zb_min,zb_max
i_MPI_Adapt, nFP_difference_max, n_itime_Adapt,n_ini_Adapt
nbfm
MPI_CartDims
MPI_AutoDomainProcs, n_procs_x0, n_procs_z0
```

There now follows a brief outline of the how the simulation is parallelised along with structural changes to the code.

### 3. CHANGES FROM SERIAL CODE

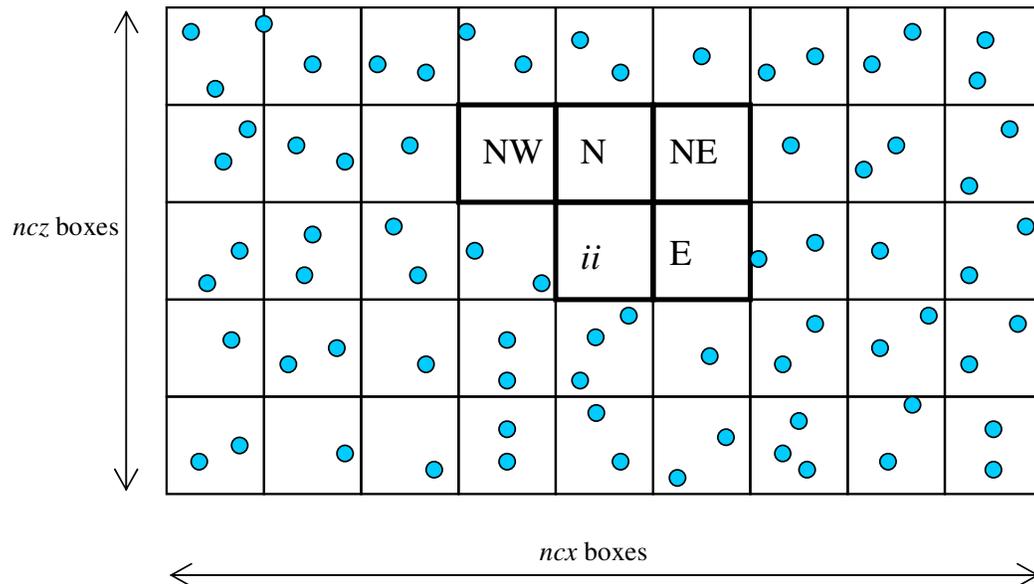
#### 3.1 Differences between the serial and parallel codes

All efforts have been made to minimise the difference between the serial and parallel codes. A number of new subroutines are necessary to deal with the transfer of particles and their information between adjacent processors detailed below. The main difference is that the MPI topology must be accessible by each subroutine (i.e. rank, east & west neighbours, MPI communicator, etc.).

Some of this has been explained in Rogers et al. (2007).

#### 3.2 Grid sweep

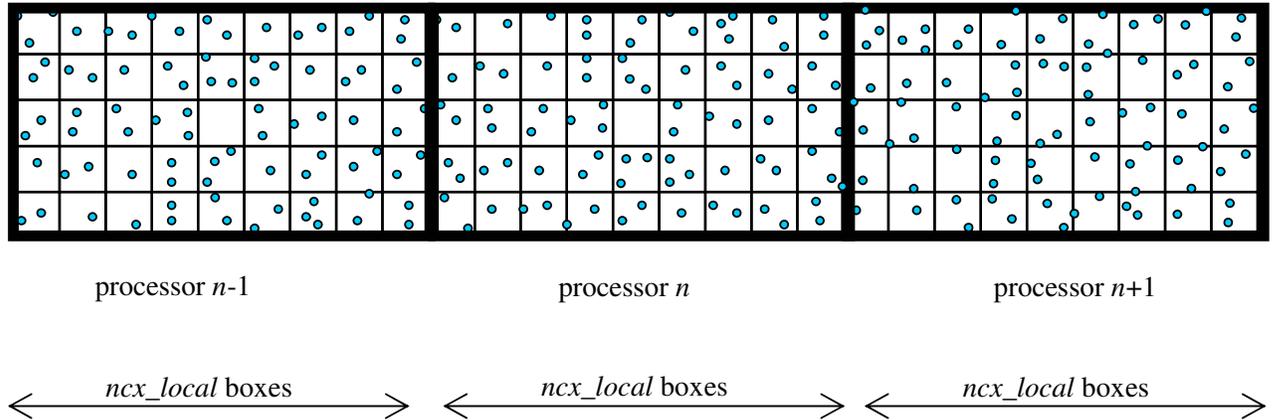
The serial SPHysics code in 2-D sweeps through the grid along the  $x$ -direction and for  $z$ -level. Around each cell, the E, N, NW & NE neighbouring boxes are checked to minimise repeating the particle interactions. This process is shown schematically in Figure 3.1.



**Figure 3.1:  $2h$  grid sweep in serial SPHysics**

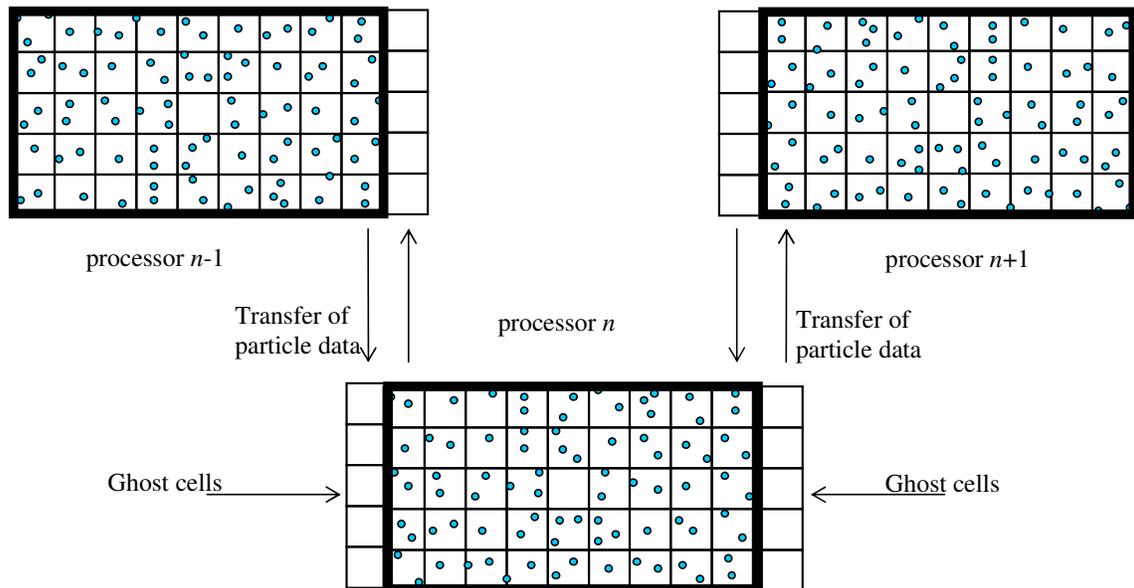
This provides an efficient method of identifying the connectivity of the particles with each other.

To parallelise the code, the workload needs to be distributed amongst the available processors. Figure 3.2 displays a typical situation where the domain has been split amongst three different processors.

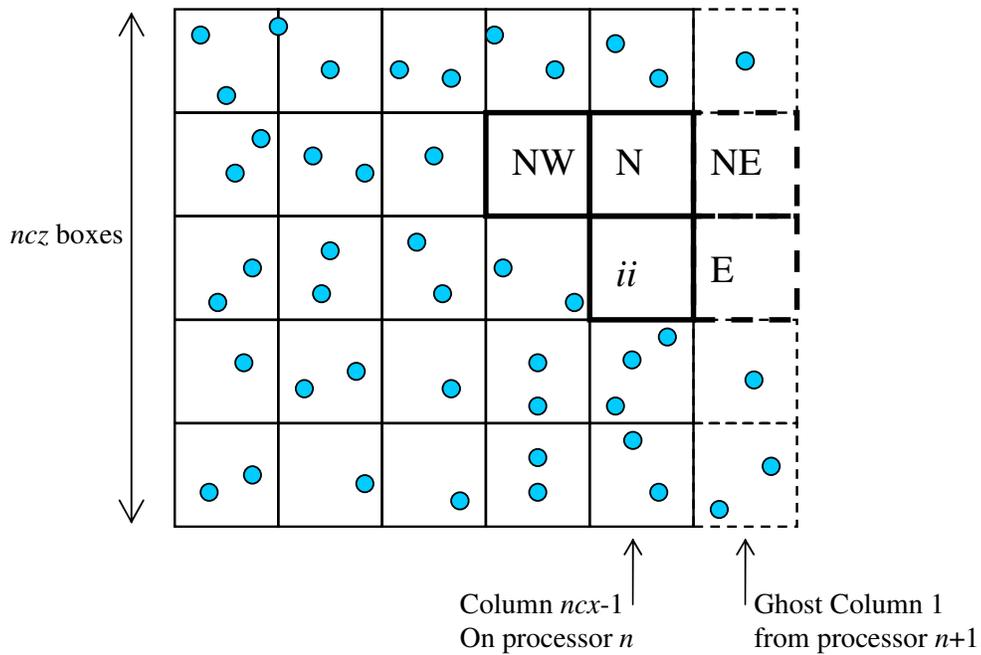


**Figure 3.2: Domain decomposition in parallel SPHysics**

However, when the domain is split up amongst several processors and box  $ii$  is located on the boundary of the processor, the code needs to know the contents of box- $ii+1$ , i.e. E & NE which lie on a different processor. The easiest method to accomplish the necessary transfer of information is to use a column of ghost cells of width  $2h$  as shown in Figure 3.3.

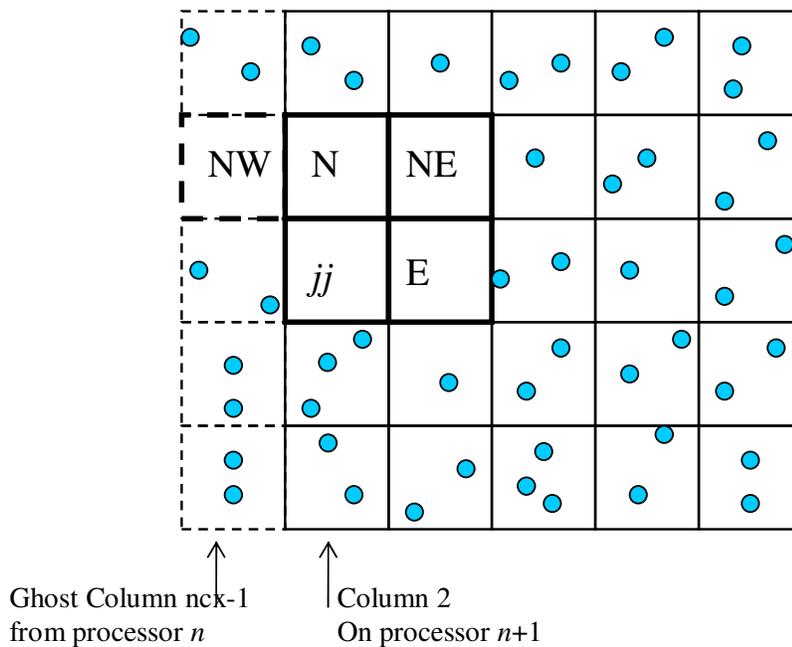


**Figure 3.3: Importing particle information into ghost cells from neighbouring processor.**



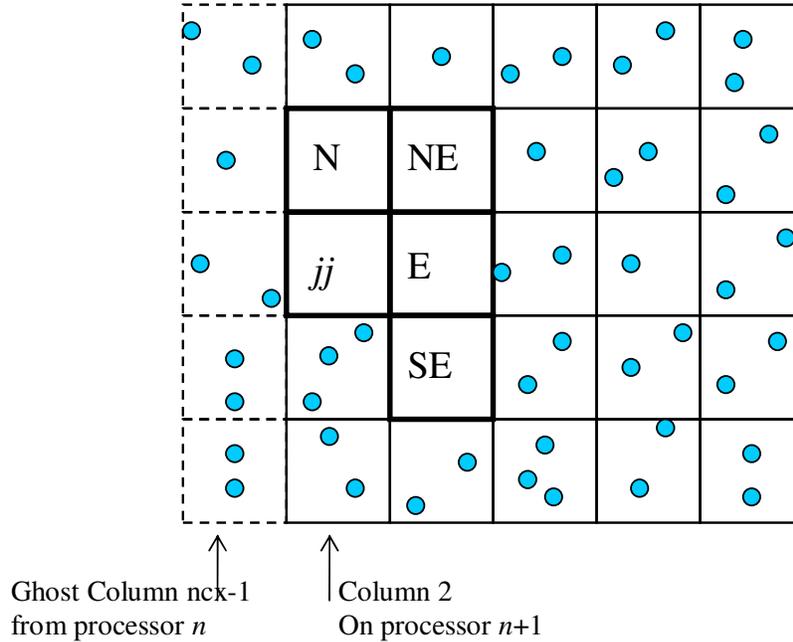
**Figure 3.4: Local neighbours of  $2h$  box  $ii$  in serial code**

Continuing with this approach means that for a  $2h$  cell,  $jj$ , on the adjacent processor  $n+1$ , all the information from column  $ncx-1$  from processor  $n$  must be imported into the ghost column 1 of processor  $n+1$  as shown in Figure 3.5.



**Figure 3.5: Local neighbours of  $2h$  box  $jj$  on boundary of rank using serial code sweeping**

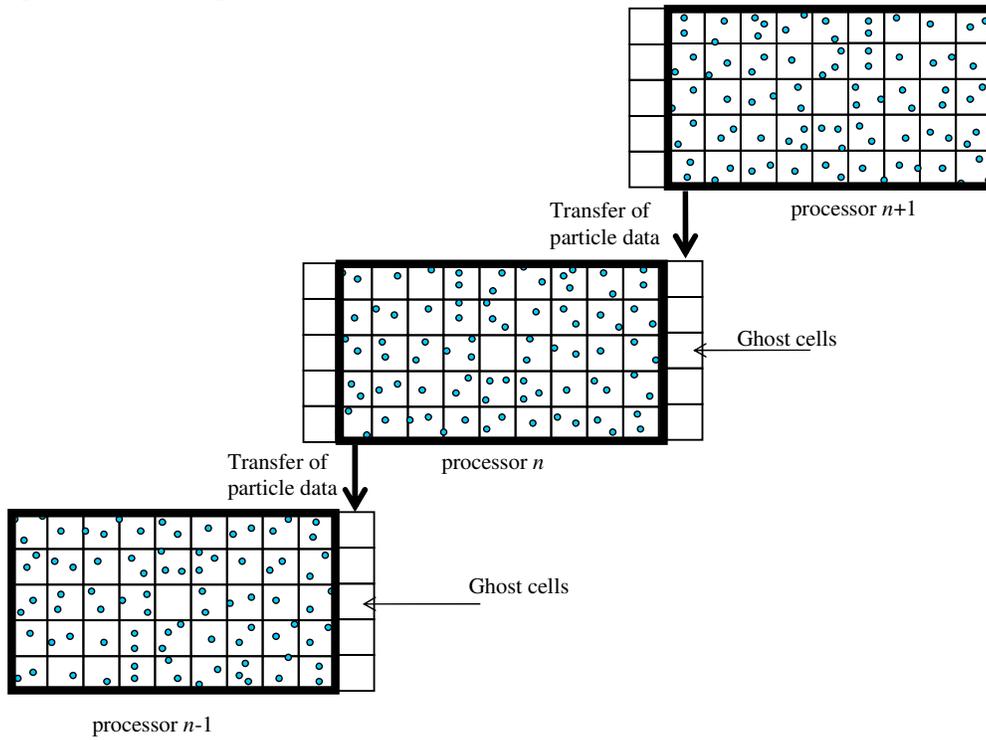
Thus, *for parallel cases whose predominant dimension is in the x-direction like many of the cases simulated in our flows* there appears to be wasted communication in the left-to-right direction since if the instead the SE cell was searched rather than the NW cell during the grid sweep as shown in Figure 3.6, then there would be no requirement to send data from processor  $n$  to processor  $n+1$  to perform the particle interactions. Only at the end of the summation process would communication be needed from processor  $n$  to processor  $n+1$  to complete the sum that has been done partly on both processes.



**Figure 3.6: Local neighbours of  $2h$  box  $ii$  in parallel code**

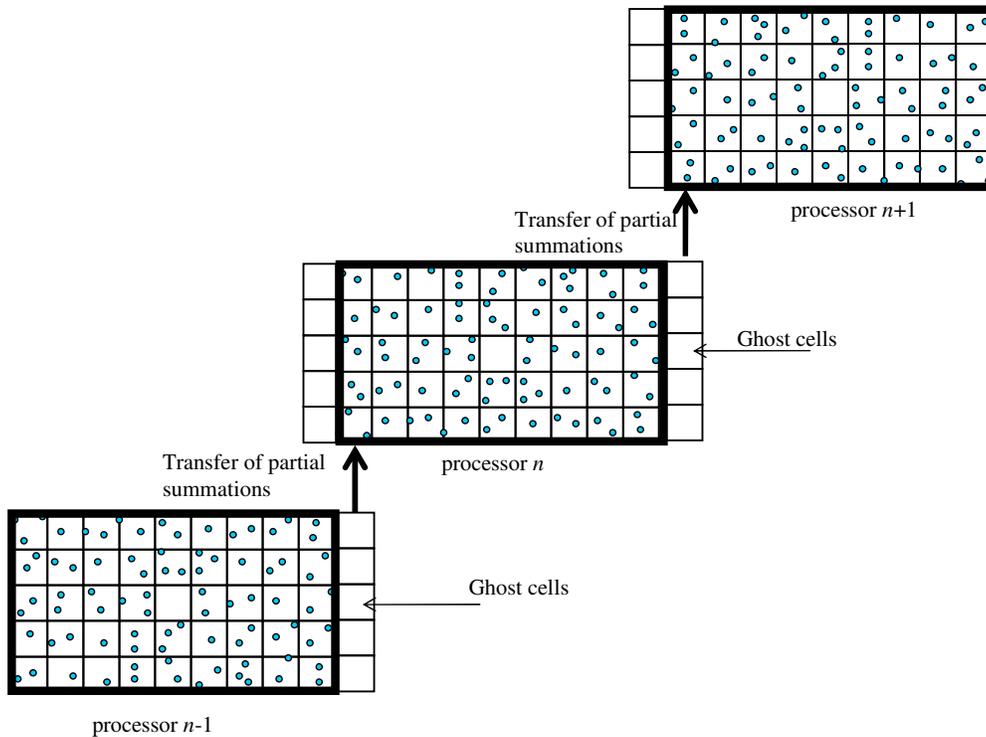
This leads to a considerable speed up. In summary, the process communication process is handled as depicted in Figure 3.7:

(i) First part of each step:



*Perform partial summations on each processor, and then ...*

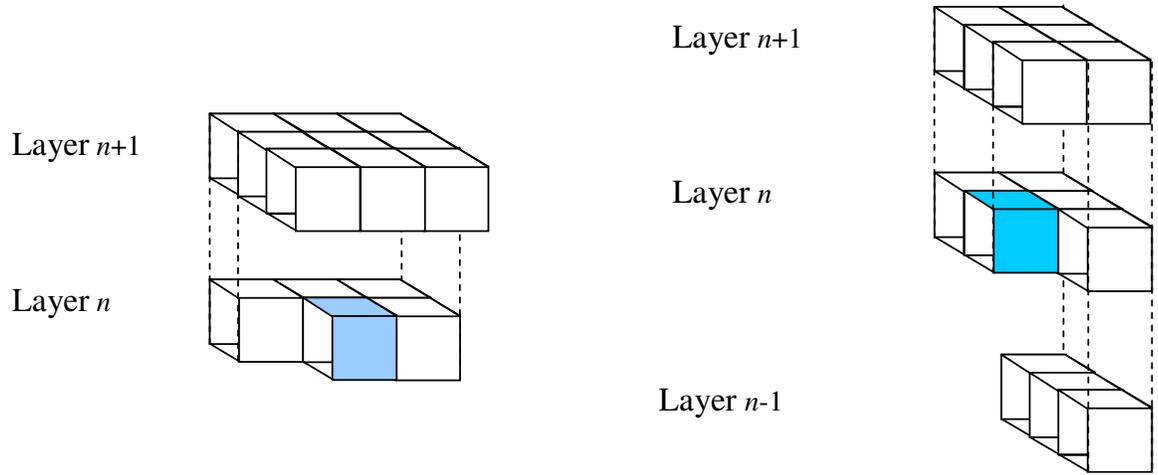
(ii) ... Second part of step



*Sum both partial summations for each processor,*

**Figure 3.7: Summation and communication procedure**

For the case of 3-D, the sweeps through the  $2h$  grid change as shown in Figure 3.8.



(a) serial code (b) parallel code  
**Figure 3.8: Change of 3-D sweeping through  $2h$  grid**

Similar operations can be performed in the North-South & Up-Down communications.

### 3.3 Differences in Naming Conventions

The subroutines most affected by parallelisation are `getdata`, `step` and `ac`. To distinguish the parallel code from the serial version, many variables that are local to each processor have the ending “`_local`” added, e.g. the number of particles `np` is a global variable, but the number of particles on a single processor is `np_local`.

All subroutines now also contain the string `_MPI` in their title to differentiate them from the serial code versions, e.g.

Serial Code subroutine: `getdata_2D.f`  
 Parallel Code subroutine: `getdata_MPI_2D.f`

### 3.4 Storage of Particle Data on each Processor: varying particle numbers

Each processor contains a different number of particles that varies over the duration of the simulation. This means that a fixed array list and dimension is impossible. This situation is made more complicated by the presence of a different number of ghost particles each timestep. Such a situation requires the use of a flexible accounting system. Rather than reordering each

timestep, which is costly and unnecessary, the straightforward approach is to use pointers. When particles leave a processor, an array keeps track of empty locations, `iarray_loc_in_use`. Then when new particles enter the processor, the spare places in the array are used before adding to the end of array using pointers: `iBPs_top_free_index`, `iFPs_top_free_index`. Guest particles which lie in the ghost cells are stored in the array space immediately after the resident particles, and hence is a temporary storage area that is overwritten each timestep.

### 3.5 Pointers

An array pointer is used to point to the correct area in memory where the particle resides. Different pointers are used for different types of particles (e.g. fluid particles (`ip_index_FP_local`)).

Each particle has been assigned a type to identify its function with a pointer as shown in Table 3.1:

Particle Type	Value of array <code>i_particleType</code>	Pointer Array	Do-loop indices:	
			Resident Particles on processor	Resident Particles + Guest Particles = Image
Stationary Boundary Particle	1	<code>ip_index_BP_local_1</code>	<code>1 : nbf_local</code>	<code>nbf_local + 1 : nbf_local_image</code>
Forced-motion Boundary Particle	2	<code>ip_index_BP_local_2</code>	<code>nbf_local+1 : nbfm_local</code>	<code>nbf_local_image+1 : nbfm_local_image</code>
Free-motion Boundary Particle	3	<code>ip_index_BP_local_3</code>	<code>nbfm_local+1 : nbfree_local</code>	<code>nbfm_local_image+1 : nbfree_local_image</code>
Fluid Particle	4	<code>ip_index_FP_local</code>	<code>nbpl_local : np_local</code>	<code>nbpl_local_image : np_local_image</code>

**Table 3.1 - Pointers used in parallel code for varying particle numbers**

### 3.6 Boundary Particles

Boundary particles represent a slightly difficult problem since while they can be considered to be disordered, their explicit connectivity needs to be known when calculating boundary surface normals. To calculate boundary surface normals requires the knowledge of the local boundary shape (which for the case of moving or deforming boundaries may not always be static or known *a priori*). Therefore, the approach taken herein was to keep an array on the root processor that stores the positions of immediately adjacent boundary particles. This list is then broadcast to all other processors. When there is a moving boundary, this list is updated

accordingly and again broadcast. This is especially important when a moving object stretches over the interface between two processors and the local connectivity needs to be known in order to calculate the new boundary normal.

### 3.7 New Subroutines

For a particle that lies within a  $2h$  box adjacent to the processor boundary, this requires summations that include particles on an adjacent processor

In order to transfer information between processors, additional subroutines have been written.

The subroutines are the same in 2-D and 3-D. Here, they are explained for 2-D.

Subroutine	Purpose
Identify_Ps_for_Exporting_toWest_MPI_2D.f Identify_Ps_for_Exporting_toBelow_MPI_2D.f	Identifies the particles that lie within $2h$ of adjacent processors
SendRecv_Data_toWest_MPI_2D.f SendRecv_Data_Conservative_toWest_MPI_2D.f SendRecv_KGC_Matrices_toWest_MPI_2D.f SendRecv_MLS_Matrices_toWest_MPI_2D.f SendRecv_filteredDensities_toWest_MPI_2D.f & SendRecv_Data_toBelow_MPI_2D.f SendRecv_Data_Conservative_toBelow_MPI_2D.f SendRecv_KGC_Matrices_toBelow_MPI_2D.f SendRecv_MLS_Matrices_toBelow_MPI_2D.f SendRecv_filteredDensities_toBelow_MPI_2D.f	Sends all the relevant data of the particles identified above to the adjacent processors
update_MLS_Summations_fromWest_MPI_2D.f update_KGC_Summations_fromWest_MPI_2D.f update_Shepard_Summations_fromWest_MPI_2D.f update_vorticity_Summations_fromWest_MPI_2D.f update_accnSummations_fromWest_MPI_2D.f update_accnSummations_Conservative_fromWest_MPI_2D.f & update_MLS_Summations_fromBelow_MPI_2D.f update_KGC_Summations_fromBelow_MPI_2D.f update_Shepard_Summations_fromBelow_MPI_2D.f update_vorticity_Summations_fromBelow_MPI_2D.f update_accnSummations_fromBelow_MPI_2D.f update_accnSummations_Conservative_fromBelow_MPI_2D.f	After performing the summations on each processor, the total summations for particles within $2h$ of adjacent processors must be collected together and summed together
update_BPInfo_onRoot_MPI_2D.f	When the boundary particles move, and possibly move from one processor to another, their connectivity on the root processor is updated and the broadcast to all other processors.
index_check_EastWest_MPI_2D.f index_check_Conservative_EastWest_MPI_2D.f & index_check_UpDown_MPI_2D.f index_check_Conservative_UpDown_MPI_2D.f	This is the key subroutine for moving the particles from one processor to another & is performed at the end of each timestep.
ipcount_to_i_MPI_2D.f	Converts a pointer to a specific index

	(needed for variable particle numbers).
Adapt_ini_MPI_2D.f	Called from <code>getdata</code> to perform initial load balancing optimization.
Adapt_MPI_Check_EastWest_MPI_2D.f & Adapt_MPI_Check_UpDown_MPI_2D.f	Called within <code>index_check</code> above, if the moving water particles are not evenly distributed across the processors, then this subroutine will change the location of the partitions between each processor
Adapt_MPI_Regrid_MPI_2D.f	Called within <code>index_check</code> above after <code>Adapt_MPI_Check</code> , this subroutine then regrids the stationary boundary particles if the location of the partition between processors changes.

For cases where only a 1-D dimensional topology is chosen, then the UpDown routines are compiled using empty subroutines, e.g. `SendRecv_Data_toBelow_NONE_MPI_2D.f` (similarly in 3-D).

### 3.8 Dynamic Load Balancing & MPI Topologies (1-D, 2-D & 3-D)

When the domain is first initialised, unless the simulation is a simple rectangle completely full of particles, the number of particles on each processor is highly unlikely to be equal. This is particularly the case for domains such as a beach where there is a sloping beach where many of the particles are concentrated offshore. Furthermore, once the simulation is in process, the particles will be moving and so the number of particles allocated to each processor will change during the simulation. Hence, some form of dynamic load balancing is required to maintain a roughly equal number of particles on each processor.

In parallelSPHysics, there is a very basic form of load balancing which occurs at two times during the simulation:

- (i) Initially, before the simulation starts
- (ii) Dynamically during the simulation

The main load balancing algorithm can be demonstrated for a 1-D Cartesian MPI topology. Let  $n_{FP}$  be the number of resident fluid particles on rank (processor)  $I$ ,  $n_{FP,East}$  be the number of resident fluid particles on the *East* neighbour rank, and  $n_{FP,West}$  the number of resident fluid particles on the *West* neighbour rank. The partitions between the ranks (or individual cores/processors) are moved according to the following conditions:

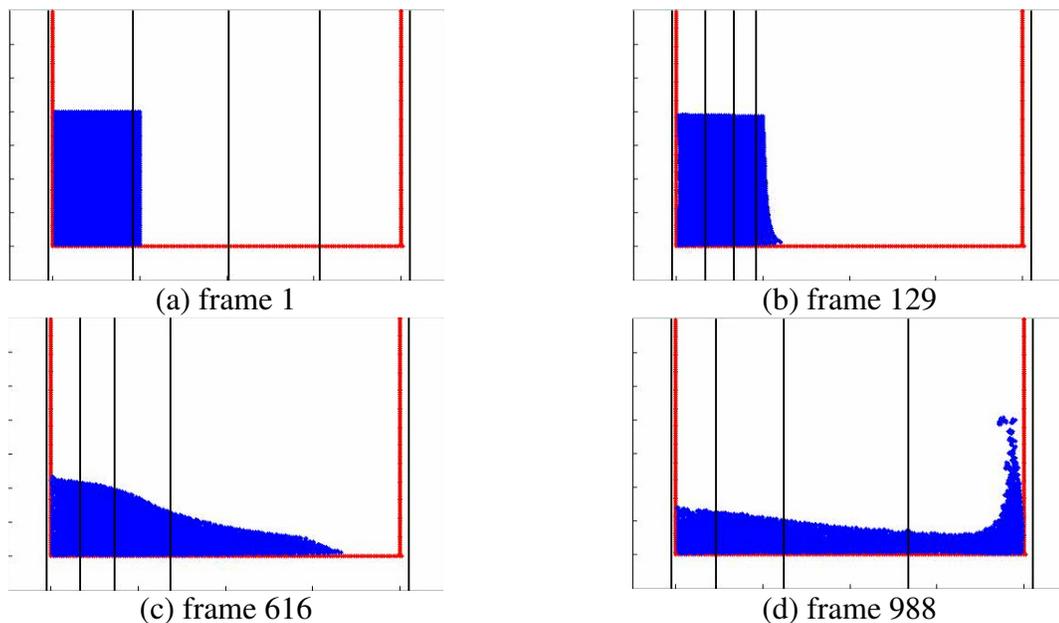
If  $(n_{FP,East} - n_{FP,West}) > n_{FP,difference\ max}$                       Move partition to the left by  $2h$

If  $(n_{FP,East} - n_{FP,West}) < -n_{FP,difference\ max}$                       Move partition to the right by  $2h$

During the simulation this procedure is repeated every `n_itime_Adapt` (default = 500) timesteps. This is not every timestep as clearly this would be a waste of computational time, checking this and performing the moving operation.

The example case shown of 2-D dam break in Figure 3.8 demonstrates that this can lead to a 40% increase in speedup. In Figure 3.8, we see the vertical black lines as the location of the partition between processors. At the beginning of the simulation, each rank (processors) have an equal length in frame 1. Then the dynamic load balancing equalises the number of particles on each processor in frame 129 before the water starts to move. In Frames 616 & 988, as the particles move rightward, the divisions between the processors moves to the right to keep the number of particles on each processor approximately equal.

The initial positions and topological information for each processor is stored in the file `MPI_DomainProcs.txt` and the domain decomposition after the initial optimisation called from subroutine `getdata` is contained in the file `MPI_Partition_init.txt`. The numbers of particles on each processor is written to a file `MPI_Partition_Positions.dat`



**Figure 3.8 Dynamic Load Balancing, black lines show spatial division between processors**

### 3.9 Choosing the Number of Processors: issues and advice

In the Case and script files there is the option to choose the MPI Cartesian topology (1, 2 or 3 dimensions) as well as the number of processors used within the simulation.

The following advice applies to both 2-D and 3-D since the code uses spatial domain decomposition. There are two main issues:

(i) There will be a limit on how many processors *should* be used; this is due to the fact that for all MPI-based codes the communication between processor cores becomes more and more important as the number of processors are increased. When moving data between processors, there are the combined issues of data travel time and synchronisation (or multi-threading). Hence due to the time spent sending and receiving data between processor cores, above a certain number of processors, there will be no gain in speedup and possibly even a reduction when the time needed for data passing becomes prohibitive.

(ii) For SPH and spatial domain decomposition, the minimum width of each sub-domain assigned to each processor core must be  $6*h$  or greater (where  $h$  = smoothing length). This is because there must be columns (or sheets) of width  $2*h$  that need to be identified as interacting with other sub-domains. Hence, if you prescribed `n_procs_x` processors in the  $x$ -direction whose minimum combined length is `n_procs_x * 6*h` and is larger than the domain length in the  $x$ -direction, then some of these processors will be wasted.

Hence, taking into account these two factors, the performance curves shown later in Figures 5.1-5.4 will not continue to show near linear speedup and efficiency for increasing numbers of processors and may even slow down.

**Advice:** For the parallelSPHysics code, the user is recommended to check the processor numbers for each coordinate direction.

**Note:** The speedup when running the code elsewhere will depend strongly on the speed of the interconnect between nodes and racks.

## 4. TEST CASES

### 4.1 Introduction

The test cases presented in this section are the same as those presented in the serial version of SPHysics but here we use resolutions that generate far larger numbers of particles.

Note: All input Case files are the same as for the serial code except with 3 lines of extra options at the end (see Section 2.6)

### 4.2 Test Case resolutions, particle numbers and Default processor numbers

#### 4.2.1 2-D Simulations

The following table summarises the number of particles for each case:

Case	Serial Code		Parallel Code		
	Resolution $\Delta x$ (m)	Number of Particles (np)	Resolution $\Delta x$ (m)	Number of Particles (np)	Default number of processors
<i>Case1.txt</i> : 2D Dam break in a box	0.03	4920	0.03	4 920**	4
<i>Case2.txt</i> : 2D Dam break evolution over a wet bottom in a box	0.005	8022	0.00125*	113 780	8
<i>Case3.txt</i> : Waves on a beach	0.01	4063	0.0005	1 368 782	64
<i>Case4.txt</i> : Tsunami generated by a sliding wedge	0.05	6604	0.0125	101 108	16
<i>Case6.txt</i> : Floating boxes in waves	0.01	5856	0.00125	338 073	32
<i>Case7.txt</i> : Focused wave group approaching trapezoid	0.02	8604	0.005	268 153	16
<i>Case8.txt</i> : Floating bodies with 2D Periodicity	0.03	5583	0.01	43 200	6

**Table 4.1 Test cases for parallelSPHysics 2D**

\* - Note `nplink_max` in `common.2D` must be changed to a value of 300 to cope with higher resolution since gate particles finish one on top of another in same  $(x, z)$  location. This may cause failure of the code to launch if the executable size is too large.

\*\* - this case is chosen purposefully to be a very small particle number so that the user can check easily that the code runs. There will be no observable speed up with such a small problem.

As mentioned earlier, the test cases are the same and therefore use the same input `CaseN.txt` files as the serial code but with two different inputs described in Section 2.6 to select the `mpif90` compiler and activate/deactivate the MPI optimisation.

#### 4.2.2 3-D Simulations

In 3-D the resolutions and numbers of particles are given in Table 4.2 below:

Case	Serial Code		Parallel Code		
	Resolution $\Delta x$ (m)	Number of Particles (np)	Resolution $\Delta x$ (m)	Number of Particles (np)	Default number of processors
<i>Case3.txt</i> : Waves on a beach	0.01	79100	0.01***	395 600	48
<i>Case4.txt</i> : Tsunami generated by a sliding wedge	0.15	20713	0.05	523 122	48
<i>Case5.txt</i> : 3-D only	0.0225	29814	0.00625	1 022 537	48
<i>Case6.txt</i> : Floating boxes in waves	0.02	11748	0.01	240 050	32
<i>Case7.txt</i> : Focused wave group approaching trapezoid	0.06	31974	0.02	690 818	36

**Table 4.2 Test cases for parallelSPHysics 3D**

\*\*\* - The domain is larger in the  $y$ -direction than for the serial code.

## 5. PERFORMANCE

### 5.1 Performance of 2-D parallelSPHysics

Here we show the speedup and efficiency of the parallelSPHysics code in 2-D. The problem used is a simple 2-D rectangular basin of length 10m, water depth 1m with  $\Delta x = 0.005\text{m}$  giving approximately 400,000 particles. The timing runs have been conducted on Itanium2 processor cores where each node is configured as a 4x Intel Itanium2 Montecito Dual Core 1.6GHz/8MB cache (i.e. 8 cores per node) and 16GB RAM. **Important:** In these results each compute node is connected via a high-bandwidth low-latency Sun Infiniband Switch which drastically reduces the communication time between nodes, i.e. not a simple ethernet connection.

In all cases, the `npar` value in file `common.2D` was modified to give the smallest possible value per processor while allowing the simulation to run.

The speedup of the code is defined by

$$S = T_1/T_N \tag{5.1}$$

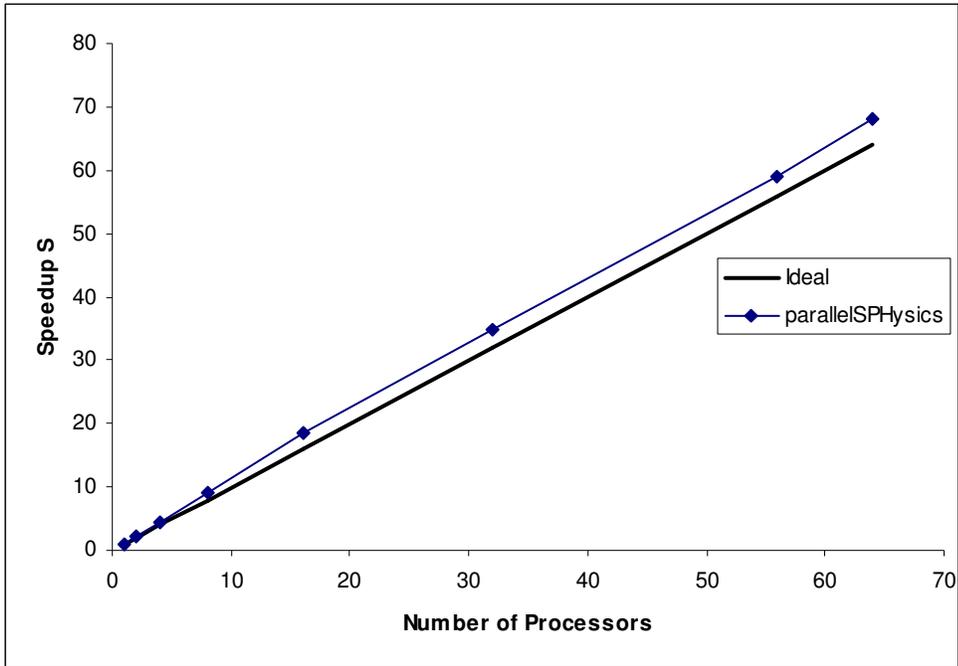
where  $T_1$  is the time taken for 1 processor and  $T_N$  is the time taken using  $N$  processors. The efficiency is defined as

$$E = S/N \tag{5.2}$$

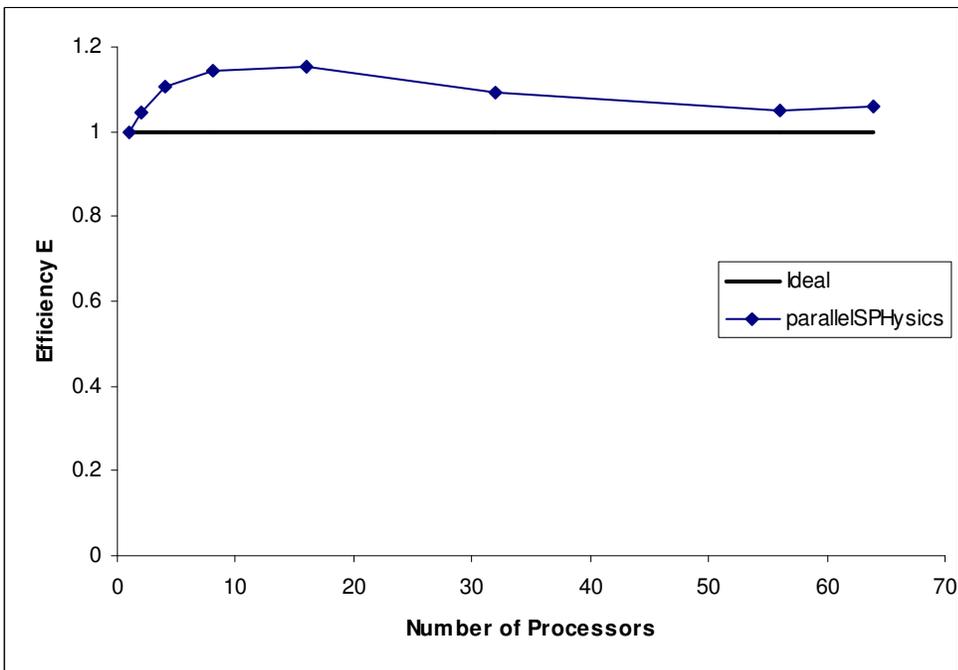
and gives a measure of that includes the redundant calculations.

Figure 5.1 shows the speedup of the 2-D code up to 64 processors where it can be seen that the code is super-linear, a behaviour which peaks around 16 processors and continues until 64 processors. This can be attributed to the optimisations made by the underlying compiler (Intel ifort). This behaviour is confirmed by the efficiency in Figure 5.2.

NOTE: Choosing the number of processors is not as simple as specifying the maximum number of processor available (see Section 3.9).



**Figure 5.1** Speedup of parallelSPHysics\_2D



**Figure 5.2** Efficiency of parallelSPHysics\_2D

## 5.2 Performance of 3-D parallelSPHysics

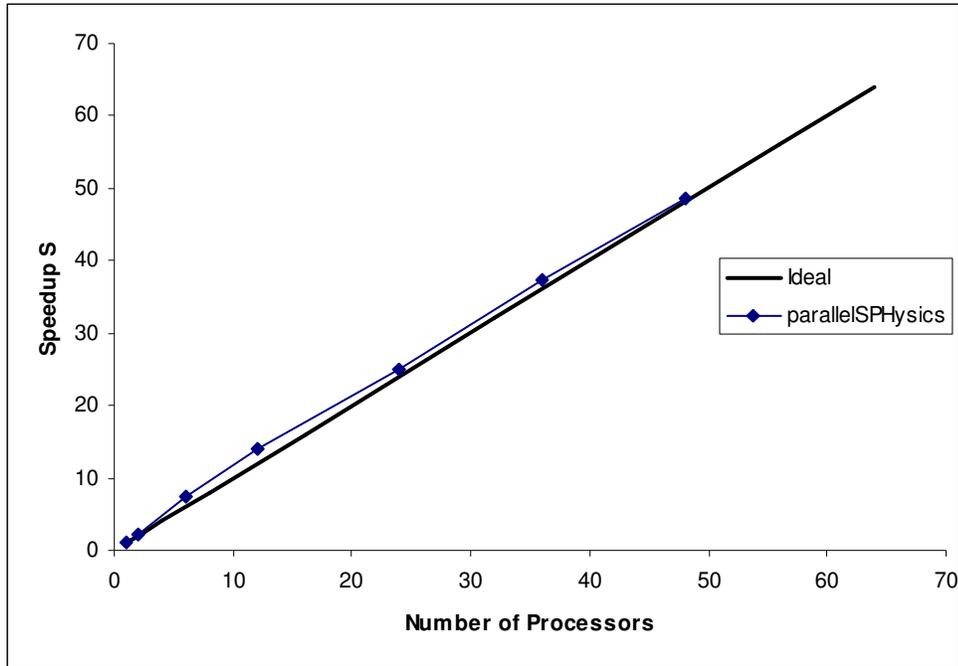
Here we show the speedup and efficiency of the parallelSPHysics code in 3-D. The problem used is a simple 3-D rectangular basin of length 10m, width 0.4m, water depth 0.25m with  $\Delta x = 0.01\text{m}$  giving approximately 1.765 million particles. The timing runs have been conducted on Itanium2 processor cores where each node is configured as a 6x Intel Itanium2 Dual Cores with 1.6GHz/8MB cache (i.e. 12 cores per node) and 16GB RAM. In all cases, the `npar` value in file `common.3D` was modified to give the smallest possible value per processor while allowing the simulation to run.

The speedup of the code and efficiency of the code are defined by Equations (5.1 & 5.2).

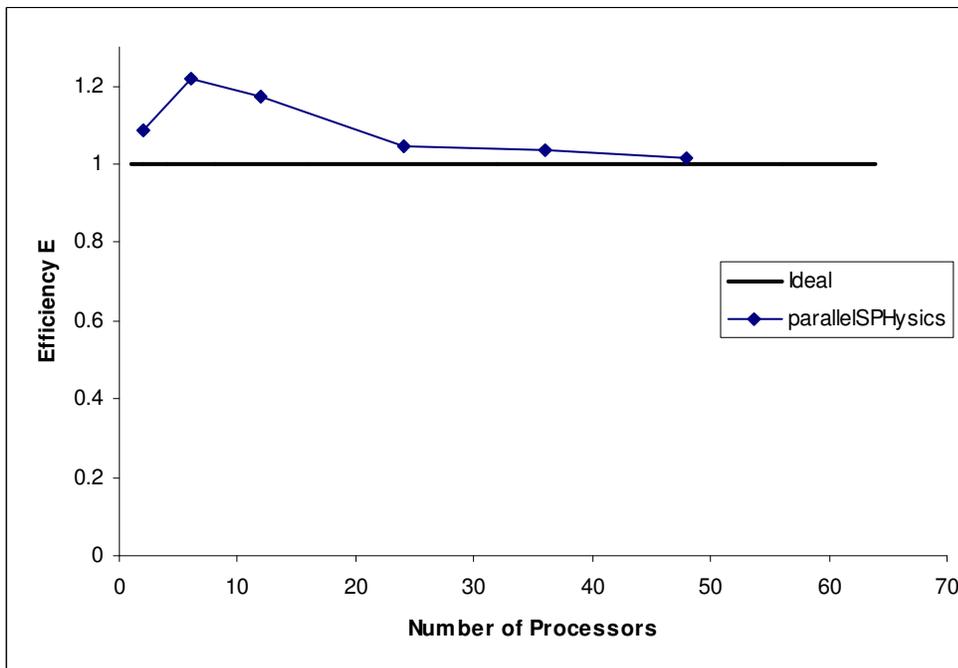
Figure 5.3 shows the speedup of the 3-D code up to 48 processors using 1-D Cartesian Topology where it can be seen that the code is super-linear, a behaviour which peaks around 16 processors and continues until 48 processors. This can be attributed to the optimisations made by the underlying compiler (gfortran) and the fast interconnect. This behaviour is confirmed by the efficiency in Figure 5.4.

As more and more processors are used, the communication becomes more important and its effect slows down the wallclock time of parallel codes. As stated in the introduction, the nature of HPC is that the best performance can only be achieved with a code optimised to individual architectures whereas our objective was a portable parallel code. Here we have not tested above 48 processors above which there is unlikely to be continued speed up.

NOTE: Choosing the number of processors is not as simple as specifying the maximum number of processor available (see Section 3.9).



**Figure 5.3** Speedup of parallelSPHysics\_3D – 1-D Cartesian Topology



**Figure 5.4** Efficiency of parallelSPHysics\_3D – 1-D Cartesian Topology

Note: these performance curves will not continue to show near linear speedup and efficiency for larger and larger numbers of processors – see discussion in Section 3.9.

## 6. VISUALISATION

To visualize the results obtained from SPHysics simulations, some basic post-processing programs have been provided in the SPHysics\_2D/Post-Processing and SPHysics-3D/Post-Processing directories.

Detailed README files, explaining the procedure to view the results using Matlab and Paraview, are available in those directories. The user is encouraged to read these README files prior to using the visualization programs.

## 7. COMPLEX GEOMETRIES

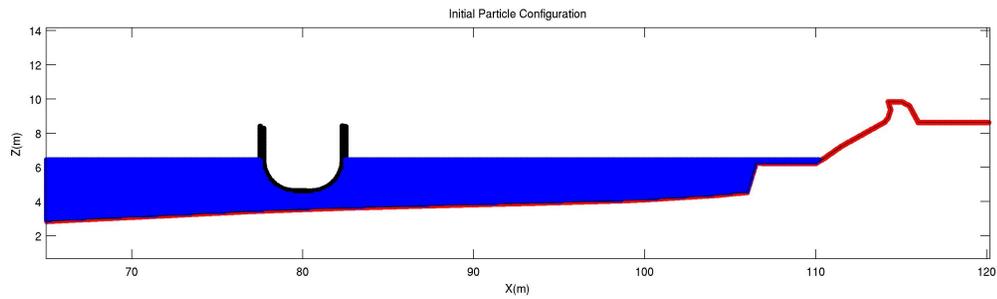
To generate complex geometries, the user has different options in 2-D and 3-D.

For 3-D, the complex geometries can be generated using the Blender software from the serial code. This functionality will be added to the 3-D parallel code in the future release 2.2)

For 2-D, there are now two new subroutines in SPHYSICSgen\_2D.f called `external_geometry` and `ComplexObject_Particles` which allow any number of boundary shapes to be imported for both fixed and free-moving objects (the functionality for moving the objects according to a predetermined motion will be added in a future release).

### 7.1 Format of input files and Example Setup

Each complex boundary is loaded in as a separate file containing only the sequential ( $x, z$ ) coordinates of the boundary seeding points. In the example `Case_withComplexBoundaries.txt`, two files are loaded in, one for the beach profile with a recurve wall (`Beach_withRecurve.txt`) and one for the ship hull in the shallow water (`ShipHull.txt`) as shown in Figure 7.1. For the free-moving objects such as the hull, the values of the object mass, moments of inertia must still be entered in the Case file.



**Figure 7.1 Complex 2-D geometry with ship hull and recurve wall**

## 8. FUTURE DEVELOPMENTS

At present the output is entirely ASCII, however, it is envisaged to add support for binary formats, along with pre- and post-processing to the code as appropriate.

## 9. REFERENCES

Gómez-Gesteira, M., Rogers, B.D., Dalrymple, R.A., Crespo, A.J.C. and Narayanaswamy, M., 2010, User Guide for the SPHysics Code v2.0, <http://www.sphysics.org>.

MPI, <http://www.mcs.anl.gov/research/projects/mpi/> and <http://www.mpi-forum.org/>

MPICH Homepage: <http://www.mcs.anl.gov/research/projects/mpich2/>

OpenMPI: <http://www.open-mpi.org/>

Rogers B.D., Dalrymple R.A., Stansby P.K., Laurence D.R.P., Development of a parallel SPH code for free-surface wave hydrodynamics, *Proc. 2<sup>nd</sup> International SPHERIC Workshop Madrid May 2007*, 111-114.

SPHysics code v2.0, <http://www.sphysics.org>.

