KARLSRUHER INSTITUT FÜR TECHNOLOGIE (KIT)

MPI-Runner for CORSIKA Parallel Simulations (Algorithms)

S. Sharma, G. Poghosyan,
Steinbuch Centre for Computing,

T. Pierog, D. Heck, J. Oehlschläger, and R. Engel
Institut für Kernphysik

Updated Version from November 18, 2011

KIT - Universität des Landes Baden-Württemberg und
nationales Forschungszentrum in der Helmholtz-Gemeinschaft

Copyright Notice

Abstract

Parallelizing the Air Shower Simulation Code CORSIKA Using Message Passing Interface (MPI): A User's Guide

CORSIKA (COsmic Ray SImulations for KAscade) is a program for detailed simulation of extensive air showers initiated by high energy cosmic ray particles. Protons, light nuclei up to iron, photons, and many other particles may be treated as primaries. The particles are tracked through the atmosphere until they undergo reactions with the air nuclei or - in the case of instable secondaries - decay. The hadronic interactions at high energies may be described by several reaction models. The MPI-Runner works to parallelize the run of CORSIKA so as to reduce the simulation time of the run and to get some performance speed up. It is construted by use of multigrid or multilevel techniques, one of the most effective ways for achieving high scalability parallel simulations on high performance computing systems.

Zusammenfassung

Parallelisierung des Luftschauer-Simulationscodes CORSIKA unter Benutzung des Message Passing Interface (MPI): Eine Benutzeranleitung

CORSIKA (COsmic Ray SImulations for KAscade) ist ein Programm für die detaillierte Simulation von ausgedehnten Luftschauern, die von hochenergetischen Teilchen der kosmischen Strahlung ausgelöst werden. Protonen, leichte Kerne bis zum Eisen, Photonen und viele andere Partikel können als Primärteilchen behandelt werden. Diese Teilchen werden durch die Atmosphäre verfolgt, bis sie Reaktionen mit den Atomen der Luft erleiden oder - im Fall instabiler Sekundärteilchen - zerfallen. Die hadronischen Wechselwirkungen bei den hohen Energien können mit verschiedenen Reaktionsmodellen beschrieben werden. Der MPI-Runner bewirkt eine Parallelisierung von CORSIKA-Rechnungen in der Art, dass die Simulationszeit eines Rechenlaufes reduziert und damit bei gleicher Leistung eine Beschleunigung erreicht wird. Er benutzt die Multi-Grid oder Multi-Level-Technik, eine der effektivsten Arten um parallele Simulationen mit hoher Skalierbarkeit auf Hochleistungs-Rechensystemen zu erreichen.

Contents

1 Algorithmic Principles of *MPI-Runner* Code

CORSIKA is a simulation code of cosmic ray interactions with the air particles. As any interaction occurs, new particles are produced and we have a new sub-shower. For every newly appeared high energy particle (energy $>$ DECTMAX) a new parallel simulation task is started using traditional sequential CORSIKA. As well for new secondary particles falling into a specified energy interval (DECTCUT $<$ energy $<$ DECTMAX), they are grouped together in an additional task to be run as a stack of particles. For that we have modeled the *MPI-Runner* that is starting, stopping, queueing, and bookkeeping the separate CORSIKA simulation tasks.

1.1 The MASTER-SLAVE Model

The MPI-Runner is a based on a classical MASTER-SLAVE model. One processor in the MPI-environment is designated as the MASTER to perform functions like scheduling, book-keeping, starting, and stopping separate simulations etc. It communicates with other processors in the environment via MPI-messages. The messages are of different types:

1. START (MASTER $\rightarrow$ SLAVE)
   To order the SLAVE to start an execution of sub-shower/particle/group.

2. REQUEST (SLAVE $\rightarrow$ MASTER)
   The SLAVE can request for the execution of new sub-shower.

3. FINISH (SLAVE $\rightarrow$ MASTER)
   The SLAVE informs/reports the master, that the execution of a particular subshower is now over.

4. STOP (MASTER $\rightarrow$ ALL SLAVES)
   To stop the execution and finalize/exit.

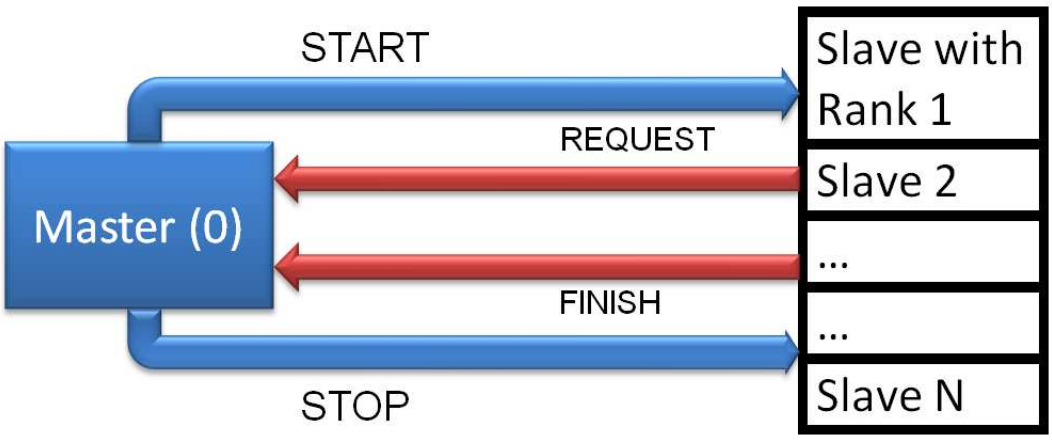

Figure 1: The Master-Slave Model.

1.2 Scheduling and Book-Keeping

1.2.1 Assigning of Individual IDs

Every job is assigned a unique ID (identification number) by the MASTER. This ID is assigned as soon as a request for a new job is received by the MASTER. Later on the MASTER can decide to START it or QUEUE it.

1.2.2 Book-Keeping

With the unique ID the MASTER can recognize every job and book-keeping is managed by writing information on to disk in the following format. The interpretation of output files is explained in the following sections.

RANK (on which) running)	Time (including Comm.)	Real Time	Cut File Name	Parent ID	Status
2	100 sec	97 sec	*.cut	1	F
3	-	-	*.cut	2	R
-	-	-	*.cut	2	Q
-	-	-	*.cut	3	E

1.2.3 Scheduling

To make decision whether a new job should run, the MASTER has to keep an account of the free RANKs/processors. The MASTER performs this scheduling by maintaining an ARRAY. The Master also keeps a QUEUE to store waiting processes. Whenever there is a request for a new process, the MASTER traverses the ARRAY. If it finds a free RANK, a START message is sent to the free RANK, otherwise the job is pushed into the QUEUE. Whenever the MASTER receives a FINISH message, a job is over and a RANK is free. The MASTER checks for jobs in the QUEUE. If there are jobs in the QUEUE, the MASTER simply reads the job from the QUEUE and orders to execute on the newly freed RANK/processor.

RANK 0 (MASTER)	1	2	3	4	5	6	7	8	...
status	1	1	1	1	0	0	1	0	...
task running (ID of task)	10	34	56	15	-	-	6	-	...
meaning of status	'1' = BUSY '0' = FREE								

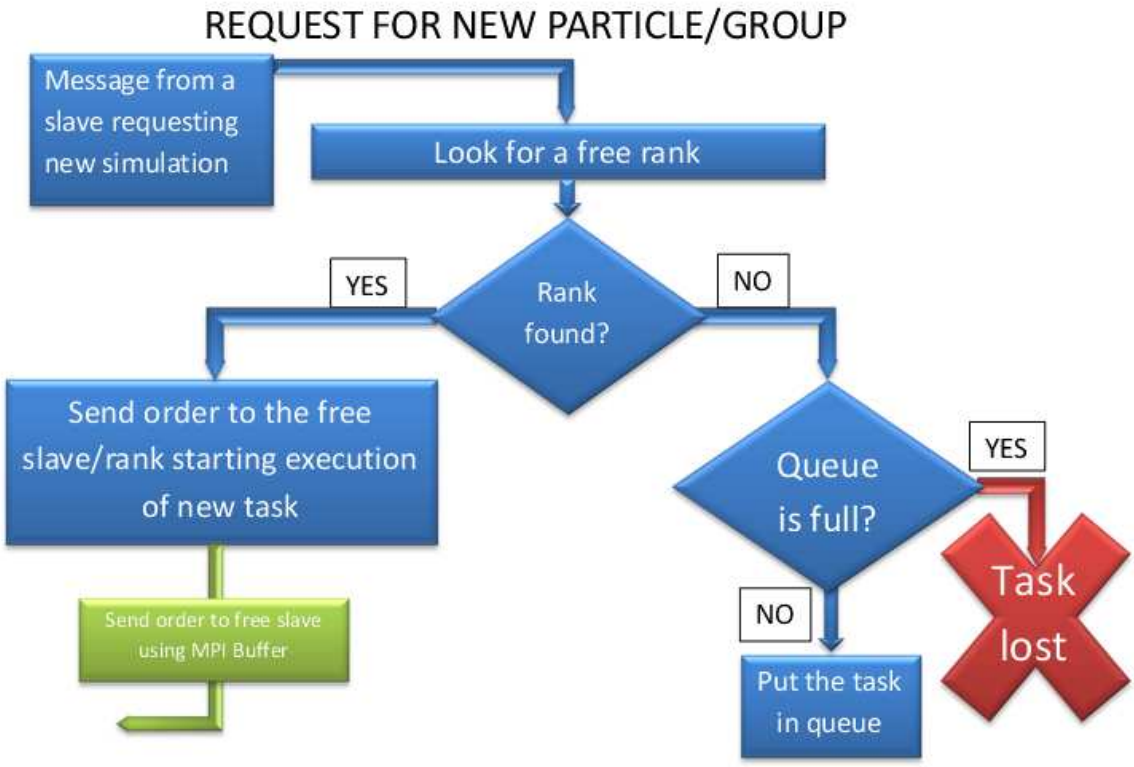

Figure 2: Scheduling of tasks.

Figure 3: Scheduling of tasks.

2 Functions

2.1 Function *main()*

In this function all the initializations are done. These initializations affect the QUEUE, the MPI buffers, the MPI environment, and the book-keeping.
The action of the *main* function depends on whether the processor is a MASTER or a SLAVE. By default the MASTER is with the RANK 0. All other RANKS are SLAVEs.

Functioning of MASTER
STEP 1:
Initialize the QUEUE.
Initialize the ARRAY (to keep track of free RANKs).
Set the unique ID to 0.
Initialize the MPI buffer.
STEP 2:
Set the MPI buffer for the primary particle.
Send the 'START' message to a SLAVE (usually the SLAVE with RANK = 1)
Update the ARRAY by setting the RANK 1 to 'BUSY'.
STEP 3:
Check if all the RANKs are 'FREE'. (call to function *check_for_finish()*)
If none of the RANK is 'BUSY', go to Step 8.
STEP 4:
Wait for any message from any SLAVE. On receiving any message, proceed to Step 5.
STEP 5:
The message is saved in an MPI buffer (say BUFF1).
If the type of message received is 'REQUEST', go to Step 6.
If the type of message received is 'FINISH', go to Step 7.
STEP 6:
If the type of message received is 'REQUEST', the job is assigned a unique ID.
Look for a free RANK (call to function *get_free_rank()*).
If a free RANK is found (say RANK n):
Send 'START' message to RANK n with the same buffer BUFF1.
Update the RANK n as 'BUSY'.
Go to Step 3.
Else
Insert the contents of the BUFF1 to the QUEUE (call to function *add()*).
Go to Step 3.
STEP 7:
If the type of message received is 'FINISH',
say that the message was sent by the RANK m.
Update the RANK m as 'FREE'.
If the QUEUE is empty:

Go to Step 3.
Else
Read the contents of one job from the QUEUE into a buffer (say BUFF2).
Send the 'START' message to the RANK m with the buffer BUFF2.
Update the RANK m as 'BUSY'.
Go to Step 3.
STEP 8:
Free the memory allocated initially.
Call to function *finish()* .

Functioning of SLAVE (SLAVE is for all RANKs other than MASTER.)
STEP 1:
Initialize the MPI buffer.
STEP 2:
Wait for any message from MASTER. On receiving any message, proceed to Step 3.
STEP 3:
The message is saved in an MPI buffer (say BUFF3).
Reconstruct the *output file name*, *cut file name* from the *run-number*, *seed*, *MPI ID*.
Example : 1, 23, 3432 maps to *DAT000001-000000023-000003432*.cut
If the type of message received is 'START', go to Step 4.
If the type of message received is 'STOP', go to Step 5.
STEP 4:
If the type of message received is 'START':
Execute subroutine *corsika()* with the same arguments as received in buffer BUFF3.
Send FINISH message to MASTER.
Go to Step 2.
STEP 5:
If the type of message received is 'STOP', finalize the MPI environment and exit.

2.2 Function *get_free_rank()*

This function is called from the *main()* (MASTER).
START
Search for a free processors from the list of RANKs.
If a free processor is found:
Returns the RANK of free processor.
Else
Returns NOT_FOUND.
STOP

2.3 Function *check_for_finish()*

This function is called from the function *main()* (MASTER).
Checks if the simulation is over, i.e., if there is no processor 'BUSY'.
START
  If simulation over:
    Returns TRUE
  Else
    Returns FALSE
STOP

2.4 Function *finish()*

This function is called from the *main()* (MASTER).
START
STEP 1:
  Send 'STOP' message to all the SLAVEs.
STEP 2:
  Finalizes the MPI environment and exit.
STOP

2.5 Subroutine *corsika()*

For the detailed functioning of the subroutine *corsika()*, please refer to the CORSIKA User's Guide.
Subroutine *corsika()* is a code written in FORTRAN. It is called by the *main()* function (SLAVE). It interacts with the MPI-Runner through the calls to the function *endoffile()*. Whenever the production of a *DAT*....cut file is complete, *corsika* calls the function *endoffile()* passing the name of the *DAT*....cut file and the information about the particles/groups in the *DAT*....cut file.

2.6 Function *endoffile(array,runnum,seed,mpiid)*

The run-number, seed, and the MPIID define the name of the *DAT*....cut file to be used by the daughter jobs. And the ARRAY defines how this *DAT*....cut file is to be read.

START
STEP 1:
  For every job, a call is made to the function *new_particle(index1,index2,runnum,seed,mpiid)*.
STOP

Example:
Let's say that CUTFILE and ARRAY contain the information about the following particles
CUTFILE:

particle number	information
1	... ...
2	... ...
3	... ...
4	... ...
5	... ...
6	... ...
7	... ...
8	... ...

ARRAY:

1	2	3	5	6	7

This shows that the CUTFILE has information about 9 particles.
ARRAY tells about the grouping (if any) of the particles.
Here, particle 1, 2 are to be simulated individually and particles 3 and 4 are grouped together. And similarly, 5 and 6 are individual tasks, finally 7 and 8 are grouped.

job	particles
1	1
2	2
3	3, 4
4	5
5	6
6	7, 8

NOTE: This shows that the size of the ARRAY gives the number of jobs (may be single particles or groups).

2.7 Function *new_particle(index1,index2,runnum,seed,mpiid)*

index1 and *index2* define the position of particles in the CUTFILE.
The run-number, seed, and MPIID define the name of the CUTFILE to be used by the daughter jobs. Therefore run-number, seed, MPIID, and the indexes define a unique job (say JOB1).

START
STEP 1:
Initialize the MPI buffer (say BUFF4).

Set the MPI buffer BUFF4 with the parameters for the JOB1.
STEP 2:
Send a ‘REQUEST’ message to the MASTER with the buffer BUFF4.
END