APPENDIX O: LS-DYNA MPP User Guide

This is a short user's guide for the MPP version of LS-DYNA. For a general guide to the capabilities of LS-DYNA and a detailed description of the input, consult the LS-DYNA User's Manual. If you have questions about this guide, find errors or omissions in it, please email https://support.ansys.com.

Supported Features

Most LS-DYNA features are available for MPP. Those that are not supported are being systematically added. Unless otherwise noted here, all options of LS-DYNA are supported by MPP/LS-DYNA.

The following are *unsupported* features:

*BOUNDARY_THERMAL_WELD

*BOUNDARY_USA_SURFACE

*CONTACT_1D

*DATABASE_AVSFLT

*DATABASE_MOVIE

*DATABASE_MPGS

*LOAD_SUPERPLASTIC_OPTION

*TERMINATION_NODE

Contact Interfaces

MPP/LS-DYNA uses a completely redesigned, highly parallel contact algorithm. The contact options currently *unsupported* include:

*CONTACT_FORCE_TRANSDUCER_CONSTRAINT

Because these options are all supported via the new, parallel contact algorithms, slight differences in results may be observed as compared to the serial and SMP versions of LS-DYNA. Work has been done to minimize these differences, but they may still be evident in some models.

For each of the supported types of contact, the keyword option MPP can be used. Adding this keyword option triggers the reading of a new control card immediately following the keyword (unless an ID card is included). See the section on *CONTACT for details of the parameters and their meanings.

Output Files and Post-Processing

For performance reasons, MPP/LS-DYNA combines many of the ASCII output files normally created by LS-DYNA into a binary format. MPP/LS-DYNA writes this output only in binary format, irrespective of the setting of the variable BINARY in *DATABASE_*OPTION*. There is a post-processing program l2a, which reads the binary database of files and produces as output the corresponding ASCII files. MPP/LS-DYNA creates the new binary files in the directory specified as the global directory in the pfile (see section pfile). The files (up to one per processor) are named binout*nnnn*, where *nnnn* is replaced by the four-digit processor number. To convert these files to ASCII, simply feed them to the l2a program like this:

```
l2a binout*
```

LS-PREPOST can read the binout files directly, so conversion is not required. We provide it for backward compatibility.

The *supported* ASCII files are:

*DATABASE_ABSTAT

*DATABASE_BNDOUT

*DATABASE_CURVOUT

*DATABASE_DEFGEO

*DATABASE_DEFORC

*DATABASE_DISBOUT

*DATABASE_ELOUT

*DATABASE_FSI

*DATABASE_FSI_SENSOR

*DATABASE_GCEOUT

*DATABASE_GLSTAT

*DATABASE_JNTFORC

*DATABASE_MATSUM

*DATABASE_NCFORC

*DATABASE_NODFOR

*DATABASE_NODOUT

*DATABASE_PBSTAT

*DATABASE_PRTUBE

*DATABASE_RBDOUT

*DATABASE_RCFORC

*DATABASE_RWFORC

*DATABASE_SBTOUT

*DATABASE_SECFORC

*DATABASE_SLEOUT

*DATABASE_SPCFORC

*DATABASE_SPHMASSFLOW

*DATABASE_SPHOUT

*DATABASE_SPHVICINITY

*DATABASE_SWFORC

*DATABASE_TRACER_ALE

*DATABASE_TPRINT

Some of the normal LS-DYNA files will have corresponding collections of files produced by MPP/LS-DYNA, with one per processor. These include the d3dump files (new names = d3dump.*nnnn*), the messag files (now mes*nnnn*) and others. Most of these will be found in the local directory specified in the pfile.

The format of the d3plot file has not been changed. It will be created in the global directory, and can be directly handled with your current graphics post-processor.

Parallel Specific Options

There are a few new command line options that are specific to the MPP version of LS-DYNA.

In the serial and SMP versions of LS-DYNA, the amount of memory required to run the problem can be specified on the command line by adding "memory=*XXX*", where *XXX* is the number of words of memory to be allocated. For the MPP code, this will result in each processor allocating *XXX* words of memory. If pre-decomposition has not been performed, one processor must perform the decomposition of the problem. This can require substantially more memory than will be required once execution has started. For this reason, there is a second memory command line option, "memory2=*YYY*." If used together with adding "memory=*XXX*," the decomposing processor will allocate *XXX* words of memory, and all other processors will allocate *YYY* words of memory.

For example, in order to run a 250,000 element crash problem on 4 processors, you might need memory=80m and memory2=20m. To run the same problem on 16 processors, you still need memory=80m, but can set memory2=6m. The value for *memory2* drops nearly linearly with the number of processors used to run the program, which works well for shared-memory systems.

See the "Memory" subsection of the "Linear Equation Solver" section in Appendix P, "Implicit Solver," for additional remarks pertaining to memory for implicit analyses.

The full deck restart capability is supported by the MPP version of LS-DYNA, but in a manner slightly different than the SMP code. Each time a restart dump file is written, a separate restart file is also written with the base name d3full. For example, when the third restart file d3dump03 is written (one for each processor, d3dump03.0000, d3dump03.0001, etc), there is also a single file written named d3full03. This d3full file (or alternately the runfull file, which is written at the interval specified in *DATABASE_BINARY_RUNRSF) is required in order to do a full deck restart in MPP instead of a d3dump or runrsf file. In order to perform a full deck restart with the MPP code, you first must prepare a full deck restart input file as for the serial/SMP version. Then, instead of giving the command line option r = d3dump03 for example, you would use the special option n = d3full03. The presence of this command line option tells the MPP code that this is a restart, not a new problem, and that the file d3full03 contains the geometry and stress data from the previous run.

PFILE

A pfile contains MPP-specific parameters that affect the execution of the program. The command line option p = pfile causes LS-DYNA to use this file. The file is split into sections, with several options in each section. Currently, the sections directory, decomposition, contact, and general are available. First, here is a sample pfile:

```
directory {
        global rundir
        local /tmp/rundir
}
contact {
        inititer 3
}
```

The file is case-insensitive and has a free format input. The sections and options currently supported are discussed below.

directory:

Holds directory-specific options.

global *path*

Path to a directory where program output should be written. If **transfer_files** is not given, this directory needs to be accessible to all processors – otherwise it is only accessed by processor 0. This directory will be created if necessary.
Default = current working directory

global_message_files

If this option appears, the message files are written in the global directory rather than the local directory
Default = disabled (message files go in the local directory)

local *path*

Path to a processor-specific local directory for scratch files. This directory will be created if necessary. If a pfile is not provided, the local directory can be defined through the environment variable LSTC_DIR_LOCAL. This directory should be on a local disk on each processor for performance reasons.
Default = global path

rmlocal

If this option, which is not available for the Windows OS, is given and **transfer_files** is active, the program attempts to clean up the **local** directories on each processor. In particular, it deletes files that are successfully transferred back to the **global** directory and removes the **local** directory if it was created. It will not delete any files if there is a failure during file copying, nor will it delete directories it did not create.
Default = disabled

repository *path*

Path to a safe directory accessible from processor 0. This directory will be created if necessary. This is intended to be used as a safekeeping/backup of files during execution and should only be used if **transfer_files** is also given. If this directory is specified, then the following actions occur:

- At program start up, any required files (d3dump, binout, etc) that cannot be located in the **global** directory are looked for in the **repository** for copying to the **local** processor directories.
- Important output files (d3dump, runrsf, d3plot, binout and others) are synchronized to the repository regularly. That is, every time one of these files is updated on the node local or the global directories, a synchronized copy is updated in the repository.

The intention is that the repository be on a redundant disk, such as NAS, to allow restarting the problem if a hardware failure should occur on the machine running the problem. It must be noted that some performance penalty must be paid for the extra communication and I/O. Effort has been made to minimize this overhead, but this option is not recommended for general use.
Default = unspecified

transfer_files

If this option is given, then processor 0 will write all output and restart files to the **global** directory (see "global" below), and scratch files to the **local** directory. All other processors will write all data to their **local** directory. At normal termination, all restart and output files will be copied from the processor specific **local** directories to the **global** directory. Also, if this is a restart from a dump file, the dump files will be distributed to the processors from the **global** directory. With this option enabled, there is no need for the processors to have shared access to a single disk for output – all files will be transferred as needed to and from the **global** directory.
Default = disabled

decomposition:

Holds decomposition-specific options.

automatic

If this option is given, an attempt is made to automatically determine a reasonable decomposition, primarily based on the initial velocity of nodes in the model. Use of the **show** option is recommended to verify a reasonable decomposition.

aledist

Distribute ALE elements to all processors.

ctcost *id1,id2,…,sf*

Apply a scale factor to the partition weight of all elements defined in the surfa and surfb sides in the list of contacts. A prefix of "a," or "b," can be added before "id1" to apply different scale factors to the surfa or surfb sides, respectively. Repeat this option as desired.

dcmem *n*

It may be in some cases that the memory requirements during the first phase of decomposition are too high. If that is found to be the case (if you get out of memory errors during decomposition phase 1), then this may provide a work around. Specifying a value *n* here will cause some routines to process the model in blocks of *n* items, when normal processing would read the whole set (of nodes, elements, whatever) all at once. This will reduce memory requirements at the cost of greater communication overhead. Most users will not need this option. Values in the range of 10,000 to 50,000 would be reasonable.

dunreflc

Time dependent load curves are usually applied to the following boundary or loading conditions on each node. By default, those curves are copied to all MPP subdomain without checking for the presence of that node in the domain or not. The curves are evaluated every cycle and may consume substantial CPU time. This command will remove curves which are not referenced in the MPP subdomain for the following keywords.

*BOUNDARY_PRESCRIBED_MOTION_NODE

*LOAD_NODE

*LOAD_SHELL_ELEMENT

*LOAD_THERMAL_VARIABLE_NODE

file_*option filename*

The name of the file that holds the decomposition information. If ***option*** is unset, this file will be created in the current working directory if it does not exist. If set, ***option*** can be entered as ***READ*** or ***WRITE***. When ***READ*** is used, LS-DYNA expects you to place the pre-decomposition file in the working directory. The job will be terminated if the file is not presented. When ***WRITE*** is used, LS-DYNA will create the file. The job will be terminated if the pre-decomposition file is in the working directory. If **file_*option*** is not specified, MPP/LS-DYNA will perform the decomposition.
Default = None

method *name*

Currently, there are two decomposition methods supported, namely *rcb* and *greedy*. Method *rcb* is Recursive Coordinate Bisection. Method *greedy* is a simple neighborhood expansion algorithm. The impact on overall runtime is problem dependent, but *rcb* generally gives the best performance.
Default = rcb

newcost

If this option appears in the decomposition section, a newer version of the element cost data is used during the decomposition with the goal of improving the load balance.

numproc *n*

The problem will be decomposed for *n* processors. If *n* > 1 and you are running on 1 processor, or if the number of processors you are running on does not evenly divide *n*, then execution terminates immediately after decomposition. Otherwise, the decomposition file is written, and execution continues. For a decomposition only run, both **numproc** and **file** should be specified.

Default = the number of processors you are running on

outdecomp *itype*

If this option appears in the decomposition section, the decomposition is written to the output file decomp_parts.lsprepost for itype = 1 or decomp_parts.ses for itype = 2. The d3plot file is not changed, and the problem will run normally.

region rx ry rz sx sy sz c2r s2r 3vec mat

See the section below on Special Decompositions for details about these decomposition options.

rcblog *filename*

This option is ignored unless the decomposition method is *rcb*. A record is written to the indicated file recording the steps taken during decomposition. This is an ASCII file giving each decomposition **region** (see the section on Special Decompositions) and the location of each subdivision for that **region**. Except for the addition of this decomposition information, the file is otherwise equivalent to the current pfile. Thus it can be used directly as the pfile for a subsequent problem, which will result in a decomposition as similar as possible between the two runs. For example, suppose a simulation is run twice, but the second time with a slightly different mesh. Because of the different meshes the problems will be distributed differently between the processors, resulting in slightly different answers due to roundoff errors. If an rcblog is used, then the resulting decompositions would be as similar as possible.

show

If this option appears in the decomposition section, the d3plot file is doctored so that the decomposition can be viewed with the post processor. Displaying material 1 will show that portion of the problem assigned to processor 0, and so on. The problem will not actually be run, but the code will terminate once the initial d3plot state has been written.

timing_start *n*

Begin timing of element calculations on cycle *n*. The appearance of this option will trigger the generation of a file named DECOMP_TIMINGS.OUT during normal termination. This file will contain information about the actual time spent doing element calculations, broken down by part.

timing_end *n*

End timing of element calculations on cycle *n*. A reasonable value is probably **timing_start** + 50 or 100.

timing_file *filename*

The file filename is assumed to be the output file DECOMP_TIMINGS.OUT from a previous run of this or a similar model. The computational cost of each part that appears in this file is then used during the decomposition instead of the built-in internal value for that part. Matching is based strictly on the user part ID. The first two lines of the file are skipped and only the first two entries on each of the remaining lines are relevant (the part ID and the cost per element).

transform_keyword

Including this option will cause global decomposition transformations to apply to all the decomposition regions created by the decomposition keywords _PARTS_-DISTRIBUTE, _PARTSET_DISTRIBUTE, _ARRANGE_PARTS, and _CONTACT_-DISTRIBUTE. If this option is not given, these regions are decomposed without any coordinate transformation applied.

vspeed

If this option is specified, a brief measurement is taken of the performance of each processor by timing a short floating point calculation. The resulting information is used during the decomposition to distribute the problem according to the relative speed of the processors. This might be of some use if the cluster has machines of significantly different speed.

contact:

This section has been largely replaced by the MPP keyword option on the normal contact card. The remaining useful options are:

alebkt *n*

Sets the bucket sort frequency for FSI (fluid-structure interaction) to once every *n* cycles.
default = 50

non_blocking

The non-blocking algorithm applies to groupable contacts. With this option, data for contact is packed/unpacked before the element loop. With this option, the contact cost is considered along with the element cost for better load balancing.

general:

Holds general options.

lstc_reduce
If this option appears, our own reduce routine is used to get consistent summation of floating point data among processors. See also, *CONTROL_MPP_IO_LSTC_-REDUCE which has the same effect.

nobeamout
Generally, whenever a beam, shell, or solid element fails, an element failure report is written to the d3hsp and message files. This can generate a lot of output. If this option appears, the element failure report is suppressed.

nodump
If this option appears, all restart dump file writing will be suppressed: d3dump, runrsf, d3full and runfull files will not be written.

nod3dump
If this option appears, writing of d3dump and runrsf files will be suppressed.

nofull
If this option appears, writing of d3full and runfull (full deck restart) files will be suppressed.

notiedio
If this option appears, the tied_nodes temp files are stored in memory instead of being output to disk. This option helps avoid problems on distributed file systems.

runrsfonly
If this option appears, writing of d3dump files will not occur – runrsf files will be written instead. Any time a d3dump *or* runrsf file would normally be written, a runrsf file will be written.

swapbytes
If this option appears, the d3plot and interface component analysis files are written in swapped byte ordering.

Special Decompositions:

These options appear in the "decomposition" section of the pfile and are only valid if the decomposition method is *rcb*. The *rcb* decomposition method works by recursively dividing the model in half, each time slicing the current piece of the model perpendicularly to one of the three coordinate axes. It chooses the axis along which the current piece of the model is longest. The result is that it tends to generate cube shaped domains aligned along the coordinate axes. This is inherent to the algorithm but is often not the behavior desired.

This situation is addressed by providing a set of coordinate transformation functions which are applied to the model before it is decomposed. The resulting deformed geometry is then passed to the decomposition algorithm, and the resulting domains are mapped back to the undeformed model. As a simple example, suppose you wanted rectangular domains aligned along a line in the *xy*-plane, 30 degrees from the *x*-axis, and twice as long along this line as in the other two dimensions. If you applied these transformations:

```
sx 0.5
rz -30
```

then you would achieve the desired effect.

Furthermore, it may be desirable for different portions of the model to be decomposed differently. It is now possible to specify different regions of the model to be decomposed with different transformations. The general form for a special decomposition would look like this:

```
decomposition {
        region { <region specifiers> <transformation> <grouping> }
        region { <region specifiers> <transformation> <grouping> }
        <transformation>
}
```

Where the region specifiers are logical combinations of **box**, **sphere**, **cylinder**, **parts**, and **silist**.
The transformation is a series of **sx**, **sy**, **sz**, **rx**, **ry**, **rz**, **c2r**, **s2r**, **3vec**, and **mat**. The grouping is either **lumped**, **together**, **nproc**, or empty. The portion of the model falling in the first region will be decomposed according to the given transformation. Any remaining part of the model in the second region will then be treated, and finally anything left over will be decomposed according to the final transformation. Any number of regions may be given, including 0. Any number of transformations may be specified. They are applied to the region in the order given.

The region specifiers are:

box xmin xmax ymin ymax zmin zmax
A box with the given extents

sphere xc yc zc r
The sphere centered at (x_c, y_c, z_c) with radius r. If r is negative, it is treated as infinite.

cylinder xc yc zc ax ay az r d

A cylinder with center at (x_c, y_c, z_c) with radius r, extending out in the direction of (a_x, a_y, a_z) for a distance of d. If d is 0, the cylinder is infinite in both directions.

parts n1 n2 n3 n4….

All parts whose user ID matches one of the given values are included in the region. Any number of values may be given.

partsets n1 n2 n3 n4….

All partsets whose user ID matches one of the given values will have all of their parts included in the region. Any number of values may be given.

silist n1 n2 n3 n4….

All elements involved in a contact interface whose user ID matches one of the given values are included in the region.

The transformations available are:

local

This is only useful in conjunction with the *CONTROL_MPP_PFILE keyword, when used inside a file included with*INCLUDE_TRANSFORM. At this point in the region processing, a transformation is inserted to invert the transformation used by *INCLUDE_TRANSFORM. The effect is that if this option appears before any other transformation options, all those options (and the subsequent decomposition) are performed in the coordinate system of the included file itself, not the global coordinate system.

sx t

Scale the current x coordinates by t.

sy t

Scale the current y coordinates by t.

sz t

Scale the current z coordinates by t.

rx t

Rotate around the current x axis by t degrees.

ry t

Rotate around the current y axis by t degrees.

rz t

Rotate around the current z axis by t degrees.

txyz x y z

Translate by (x, y, z).

mat m11 m12 m13 m21 m22 m23 m31 m32 m33

Transform the coordinates by matrix multiplication:

$$\text{Transformed Coordinates} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

3vec v11 v12 v13 v21 v22 v23 v31 v32 v33

Transform the coordinates by the inverse of the transpose matrix:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} v_{11} & v_{21} & v_{31} \\ v_{12} & v_{22} & v_{32} \\ v_{13} & v_{23} & v_{33} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$
$$= \mathrm{V}^{\mathrm{T}} \times \text{transformed coordinates}$$

This appears complicated, but in practice is very intuitive: instead of decomposing into cubes aligned along the coordinate axes, *rcb* will decompose into parallelepipeds whose edges are aligned with the three vectors (v_{11}, v_{12}, v_{13}), (v_{21}, v_{22}, v_{23}), and (v_{31}, v_{32}, v_{33}). Furthermore, the relative lengths of the edges of the decomposition domains will correspond to the relative lengths of these vectors.

C2R x0 y0 z0 vx1 vy1 vz1 vx2 vy2 vz2

The part is converted into a cylindrical coordinate system with origin at (x_0, y_0, z_0), cylinder axis (v_{x1}, v_{y1}, v_{z1}) and $\theta = 0$ along the vector (v_{x2}, v_{y2}, v_{z2}). You can think of this as tearing the model along the (v_{x2}, v_{y2}, v_{z2}) vector and unwrapping it around the (v_{x1}, v_{y1}, v_{z1}) axis. The effect is to create decomposition domains that are "cubes" in cylindrical coordinates: they are portions of cylindrical shells. The actual transformation is:

$$\text{new}(x, y, z) = \text{cylindrical coordinates}(r, \theta, z)$$

Knowing the order of the coordinates is important if combining transformations, as in the example below.

S2R x0 y0 z0 vx1 vy1 vz1 vx2 vy2 vz2

Just like the above, but for spherical coordinates. The (v_{x1}, v_{y1}, v_{z1}) vector is the $\phi = 0$ axis.

$$\text{new}(x, y, z) = \text{spherical coordinates}(\rho, \theta, \phi)$$

The grouping qualifier is:

lumped

Group all elements in the region on a single processor. If this qualifier is not given, the elements in the region are distributed across all processors.

together

Group all elements in the region as one "super" element and join decomposition together with other remaining parts. This qualifier works like "lumped" to keep all elements in the region on a single processor. It also assigns them on the same processor with their neighboring elements with load balanced. In general, this qualifier is recommended over "lumped."

nproc *n frstp*

This region will be distributed to *n* processors which usually is a subset of **numproc** and starting from rank **frstp**. If **frstp** is not given, the code will select the *n* least costly processors from **numproc** automatically.

Examples:

rz 45

will generate domains rotated -45° around the *z*-axis.

C2R 0 0 0 0 0 1 1 0 0

will generate cylindrical shells of domains. They will have their axis along the vector (0,0,1), and will start at the vector (1,0,0). Note that the part will be cut at (1,0,0), so no domains will cross this boundary. If there is a natural boundary or opening in your part, the $\theta = 0$ vector should point through this opening. Note also that if the part is, say, a cylinder 100 units tall and 50 units in radius, after the C2R transformation the part will fit inside the box: $x = [0,50]$, $y = [0, 2\pi)$, $z = [0,100]$. In particular, the new y coordinates (θ) will be very small compared to the other coordinate directions. It is therefore likely that every decomposition domain will extend through the complete transformed y-direction. This means that each domain will be a shell completely around the original cylinder. If you want to split the domains along radial lines, try this pair of transformations:

C2R 0 0 0 0 0 1 1 0 0
SY 5000

This will do the above C2R, but then scale y by 5000. This will result in the part appearing to be about 30,000 long in the y-direction, long enough that every decomposition domain will divide the part in this (transformed) y-direction. The result will be decomposition domains that are radial "wedges" in the original part.

General combinations of transformations can be specified, and they are applied in order:

SX 5 SY .2 RZ 30

will scale x, then y, and then rotate.

A more general decomposition might look like:

```
decomposition { rx 45 sz 10
    region { parts 1 2 3 4 5 and sphere 0 0 0 200 lumped }
    region { box 0 100 –1.e+8 1.e+8 0 500 or sphere 100 0 200 200 rx 20 }
}
```

This would take elements that have user ID 1, 2, 3, 4, or 5 for their part *and* that lie in the sphere of radius 200 centered at (0,0,0), and place them all on one processor.

Then, any remaining elements that lie in the given box *or* the sphere of radius 200 centered at (100,0,200) would be rotated 20° in the *x*-direction and then decomposed across all processors. Finally, anything remaining would be rotated 45° in the *x*-direction, scaled 10 in the *z*-direction, and distributed to all processors. In general, region qualifiers can be combined using the logical operations *and*, *or*, and *not*. Grouping using parentheses is also supported.

Execution of MPP/LS-DYNA

MPP/LS-DYNA runs under a parallel environment provided by the hardware vendor. The execution of the program therefore varies from machine to machine. On some platforms, command line parameters can be passed directly on the command line. For others, the use of the names file is required. The names file is supported on all systems.

The serial/SMP code supports the use of the SIGINT signal (usually Ctrl-C) to interrupt the execution and prompt for user input, generally referred to as "sense switches." The MPP code also supports this capability. However, on many systems a shell script or front end program (generally "mpirun") is required to start MPI applications. Pressing Ctrl-C on some systems will kill this process, and thus kill the running MPP-DYNA executable. As a workaround, there are two ways to issue this sense-switch:

1. Create a file called D3KIL with the desired sense switch in it. LS-DYNA checks for this file every 100 cycles and executes this sense-switch.

2. When the MPP code begins execution, it creates a file bg_switch in the current working directory. This file contains the following single line:

   ```
   ssh -x <machine name> kill -INT <PID>
   ```

   where < machine name > is the hostname of the machine on which the root MPP-DYNA process is running, and <PID> is its process ID. Thus, simply executing this file will send the appropriate signal and prompt the user to enter the desired sense switch. Alternatively, a file with name switch can be created where the desired sense switch is included in the file. When bg_switch is executed, the desired 'sense switch' is applied.

Here is a simple table to show how to run the program on various platforms. Of course, scripts are often written to mask these differences.

Platform	Execution Command
DEC Alpha	dmpirun –np n mpp-dyna
Fujitsu	jobexec –vp n –mem m mpp-dyna
Hitachi	mpirun –np n mpp-dyna
HP	mpp-dyna –np n
IBM	#!/bin./ksh export MP_PROC=n export MP_LABELIO=no export MP_EUILIB=us export MPI_EUIDEVICE=css0 poe mpp-dyna
NEC	mpirun –np n mpp-dyna
SGI	mpirun –np n mpp-dyna
Sun	tmrun –np n mpp-dyna

In the above, **n** is the number of processors, **mpp-dyna** is the name of the MPP/LS-DYNA executable, and **m** is the MB of real memory.