Implicit Data Region

There is an implicit data region in each subprogram that starts before the first executable statement and ends after the last executable statement.

Declarative Data Directive

A declarative data directive is used to specify that data is to be allocated in device memory for the duration of the implicit data region of the subprogram.

C

```
#pragma acc clause [[,] clause]… new-line
#pragma acc function( name ) clause [[,] clause]… new-line
```

Fortran

```
!$acc clause [[,] clause]…
```

clauses

```
copy( list )
copyin( list )
copyout( list )
local( list )
deviceptr( list )
device resident( list )
mirror( list )
reflected( list )
```

Update Directive

An update directive copies data from host memory to data already allocated in device memory, or vice versa. An update directive may appear in any data region including in an implicit data region.

C

```
#pragma acc update clause [[,] clause]… new-line
```

Fortran

```
!$acc update clause [[,] clause]…
```

clauses

```
host( list )
device( list )
async [(handle)]
```

Device Present Directive

A device present directive finds an existing device copy of data allocated in a data region surrounding the call to this procedure.

C

```
#pragma acc device present( list )
```

Fortran

```
!$acc device present( list )
```

Compiler Options

-tp=targetaccelerator

Enable accelerator directives with the option `-ta=<targetaccelerator>[:<suboptions>]…`

where `<targetaccelerator>` can be

`host`: compiles for host-only execution
`nvidia`: comp iles for NVIDIA GPUs
`host,nvidia`: PGI Unified Binary to execute on either the host or an NVIDIA GPU

and `<suboptions>` can be

`cc10`, `cc11`, `cc12`, `cc13`, `cc20`: Generate code for compute capability 1.0, 1.1, 1.2, 1.3 or 2.0; multiple selections are valid.
`cuda2.3`, `cuda3.0`, `cuda3.1`, `cuda3.2`, `cuda4.0`: Specify which NVIDIA toolkit version to use.
`fastmath`: Use fast (but lower precision) math routines.
`keepbin`: Keep the intermediate CUDA binary files.
`keepptx`: Keep the intermediate CUDA PTX files.
`keepgpu`: Keep the intermediate CUDA GPU source files.
`time`: Link in a simple kernel profiling library.

Example: `-ta=host,nvidia:cc13,time`

-tp=targetprocessor[,targetprocessor]…

Selects which x86 host processor(s) for which to compile.

Example: `-tp=nehalem-64,barcelona-64`

-Minfo or -Minfo=accel

Prints compiler informational messages.

-Minfo=ccff

Save the compiler informational messages in the executable for use by PGPROF.

Profiling

```
% pgcollect a.out
```

Executes the program under control of a sampling runtime profiler. Generates a file named `pgprof.out` containing profile data.

```
% pgprof pgprof.out -exe a.out
```

PGI®
www.pgroup.com/accelerate
sales@pgroup.com

PGI Accelerator™ Quick Reference Card

The PGI Accelerator programming model includes a collection of compiler directives to specify regions of code in standard Fortran and C programs that can be offloaded from a *host* CPU to an attached *accelerator*, providing portability across operating systems and various types of host CPUs and accelerators. The most fundamental PGI Accelerator directive is a region directive, which declares a compute region or data region that applies to the immediately following structured block. A structured block is a single statement or compound statement in C, or a sequence of statements in Fortran with a single entry at the top and a single exit at the bottom.

PGI Accelerator Directive Syntax

Only one directive-name per directive statement. Clause order is not significant and may be repeated unless otherwise specified. Where applicable, clause list arguments are comma-separated variable names, array names, or subarrays with subscript ranges..

C

```
#pragma acc directive-name [clause [[,] clause]...] new-line
```

Fortran

```
!$acc directive-name [clause [[,] clause]...]
```

Compute Region Directive

A compute region directive surrounds the loops to execute on the accelerator device.

C

```
#pragma acc region [clause [[,] clause]…] new-line
      { structured block }
```

Fortran

```
!$acc region [clause [[,] clause]…]
      structured block
!$acc end region
```

data clauses

```
copy( list )
copyin( list )
copyout( list )
local( list )
deviceptr( list )
update device( list )
update host( list )
```

list is a list of variables or arrays; see the Data Clauses section.

other clauses

`if( condition )` When the condition is nonzero or true, the compute region will execute on the accelerator; otherwise, the region will execute on the host.
`async[( handle )]`
The region will execute asynchronously with host computation; a handle may be specified for use in a wait directive or `acc_async_wait` call.

Data Region Directive

A data region directive defines a region of the program within which data is replicated on the GPU.

C

```
#pragma acc data region clause [[,] clause]… new-line
      { structured block }
```

Fortran

```
!$acc data region clause [[,] clause]…
      structured block
!$acc end data region
```

data clauses

```
copy( list )
copyin( list )
copyout( list )
local( list )
deviceptr( list )
mirror( list )
update device( list )
update host( list )
```

Wait Directive

A wait directive causes the program to wait until all asynchronous activities associated with the given handle are complete, or until all asynchronous activities are complete if no handle is specified.

C

```
#pragma acc wait [(handle)]
```

Fortran

```
!$acc wait [(handle)]
```

Environment Variables

ACC_DEVICE *device*
Specifies whether to run the accelerator regions compiled with PGI Unified Binary™ on the host or on an NVIDIA GPU. Valid values are **nvidia** or **host**. Override in the program with a call to **acc_set_device**.

ACC_DEVICE_NUM *num*
Specifies which device number to use. Override in the program with a call to **acc_set_device_num**.

ACC_NOTIFY *num*
If the value is nonzero, a single line will be printed to **stderr** for every accelerator kernel launched.

Conditional Compilation

The _ACCEL preprocessor macro is defined to have value yyyymm when compiled with PGI Accelerator directives enabled. The version described here is 201011.

Data Clauses

The description applies to the clauses used on compute regions, data regions, and to the standalone declarative data directives and executable update directives.

copy(*list* **)**
Allocates the data in *list* on the device and copies the data from the host to the device when entering the region,and copies the data from the device to the host when exiting the region.

copyin(*list* **)**
Allocates the data in *list* on the device and copies the data from the host to the device when entering the region.

copyout(*list* **)**
Allocates the data in *list* on the device and copies the data from the device to the host when exiting the region.

local(*list* **)**
Allocates the data in *list* on the device, but does not automatically copy the data between the host and device.

deviceptr(*list* **)**
C only; the list entries must be pointer variables that contain device addresses, such as from **cudaMalloc**.

device resident(*list* **)**
Declares that the data in list are to be allocated only on the device, and are accessible only within compute regions.

update device(*list* **)**
Copies the data from the host to the already-allocated space on the device for the data in *list*, when entering a nested data region, compute region, or at the update directive.

update host(*list* **)**
Copies the data from the already-allocated space on the device to the host for the data in *list*, when exiting a nested data region, compute region, or at the update directive.

mirror(*list* **)**
Fortran-only. Specifies that the data in the *list* should mirror the allocation on the host; does not automatically copy data.

reflected(*list* **)**
Data in *list* must be dummy arguments. Specifies that data in *list* must already be allocated on the device by the caller.

Loop Scheduling Directive

A loop scheduling directive applies to the immediately following loop, and describes the loop parallelism to use when mapping the loop for execution on the accelerator.

C

#pragma acc for [clause [[,] clause]…] new-line

Fortran

!$acc do [clause [[,] clause]…]

clauses

host [(*width***)]**
parallel [(*width***)]**
seq [(*width***)]**
vector [(*width***)]**
unroll(*factor* **)**
independent
private(*list* **)**
cache(*list* **)**

loop mapping clauses

The loop mapping clauses specify the mapping of loop-level parallelism onto the accelerator parallelism. Multiple clauses may be specified on a single loop, in which case all but one *must* have a width clause.

host [(*width***)]**
Execute the whole loop or *width* iterations of the loop on the host.

parallel [*(width)***]**
Execute the loop in parallel on all cores, or at most *width* cores.

seq [*(width)***]**
Execute the loop sequentially on the device.

vector [*(width)***]**
Execute the loop in SIMD or vector mode with vectors of size *width*.

unroll(*factor* **)**
Unroll *factor* iterations of the parallel, sequential or vector loop. May appear after a parallel, seq, and/or vector clause.

other loop clauses

independent
Specifies that the loop iterations are data-independent, and can be executed in parallel.

private(*list* **)**
Allocates one copy of the data in *list* for each iteration of the loop.

cache(*list* **)**
Prioritizes the data in *list* for placement in the highest level of the data cache.

Runtime Routines

Prototypes or interfaces for the runtime library routines, along with datatypes and enumeration types, are available as follows.

C

```
#include "accel.h"
```

Fortran

```
use accel_lib or #include "accel_lib.h"
```

acc_get_num_devices(*devicetype* **)**
Returns the number of devices of the specified type; the only valid device type is **acc_device_nvidia.**

acc_set_device(*devicetype* **)**
Sets the device type to use to execute compute regions; this assumes that the code has been compiled with PGI Unified Binary. The valid device types are **acc_device_nvidia** or **acc_device_host**.

acc_get_device()
Returns the device type that will be used to execute the next accelerator region.

acc_set_device_num(*devicenum, devicetype* **)**
Sets the device number of the given type to use to execute accelerator regions.

acc_get_device_num(*devicetype* **)**
Returns the device number of the given type that will be used to execute the next accelerator compute region.

acc_async_test(*handle* **)**
Returns nonzero or .TRUE. if all asynchronous activities associated with the given handle have been completed; returns zero or .FALSE. otherwise.

acc_async_wait(*handle* **)**
Waits until all asynchronous activities associated with the given handle have been completed.

acc_init(*devicetype* **)**
Initializes the runtime library for that device type; the default is to initialize when the first accelerator region is entered.

acc_shutdown(*devicetype* **)**
Disconnects the program from the accelerator device; the next accelerator region will have to reinitialize the device.

acc_on_device(*devicetype* **)**
This is used in accelerator compute regions to take different execution paths depending on whether the program is running on an accelerator or on the host. The argument must be a compile-time constant.