Unified Parallel C (UPC) User Guide

007–5604–004

New Features in This Manual

This rewrite of the *Unified Parallel C (UPC) User Guide* supports the SGI Performance Suite 1.4 release.

Major Documentation Changes

Performed the following:

- Added "OFED Configuration for UPC" on page 4.
- Updated "UPC Quick Start on SGI UV or SGI ICE Systems" on page 7.

Record of Revision

Version	Description
001	April 2010 Original Printing.
002	June 2010 Updated to support the SGI ProPack 7 Service Pack 1 release.
003	Nobember 2011 Updated to support the SGI Performace Suite 1.3 release.
004	May 2012 Updated to support the SGI Performace Suite 1.4 release.

Contents

About This Manual

This publication documents the SGI implementation of the Unified Parallel C (UPC) parallel extension to the C programming language standard.

Obtaining Publications

You can obtain SGI documentation in the following ways:

- See the SGI Technical Publications Library at: http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, release notes, man pages, and other information.
- You can also view man pages by typing `man` *title* on a command line.

Related Publications and Other Sources

Material about UPC is available from a variety of sources. Some of these, particularly webpages, include pointers to other resources. Following is a list of these sources:

- *UPC: Distributed Shared Memory Programming*

  Authors: Tarek El-Ghazawi, William Carlson, Thomas Sterling, Katherine Yelick; ISBN: 0-471-22048-5 ; Published by John Wiley and Sons- May, 2005

- http://upc.gwu.edu

  Contains much information that is relevant to UPC.

- http://upc.gwu.edu/docs/upc_specs_1.2.pdf

  Contains the description of Version 1.2 of the UPC programming language.

- http://upc.gwu.edu/downloads/Manual-1.2.pdf

  Contains a discussion about the UPC language features.

- `sgiupc`(1) man page

  SGI Unified Parallel C (UPC) compiler man page describes the `sgiupc`(1) command. `sgiupc` is the front-end to the SGI UPC compiler suite. It handles all stages of the UPC compilation process: UPC language preprocessing, UPC-to-C translation, back- end C compilation, and linking with UPC runtime libraries.

- *Message Passing Toolkit (MPT) User Guide*

  Describes industry-standard message passing protocol optimized for SGI computers.

- *MPInside Reference Guide*

  Documents the SGI MPInside MPI profiling tool.

- *SGI UV GRU Development Kit Programmer Guide*

  Documents the SGI UV global reference unit (GRU) development kit. It describes the application program interface (API) that allows direct access to GRU functionality.

- *SGI UV 2000 System User Guide*

  This guide provides an overview of the architecture and descriptions of the major components that compose the SGI UV 2000 system. It also provides the standard procedures for powering on and powering off the system, basic troubleshooting information, and important safety and regulatory specifications.

- *SGI Altix UV 1000 System User's Guide*

  This guide provides an overview of the architecture and descriptions of the major components that compose the SGI UV 1000 system. It also provides the standard procedures for powering on and powering off the system, basic troubleshooting information, and important safety and regulatory specifications.

- *SGI Altix UV 100 System User's Guide*

  This guide provides an overview of the architecture and descriptions of the major components that compose the SGI UV 100 system. It also provides the standard procedures for powering on and powering off the system, basic troubleshooting information, and important safety and regulatory specifications.

- *SGI Altix ICE 8200 Series System Hardware User's Guide*

  Describes the features of the SGI ICE 8200 series systems and provides operating instructions and general troubleshooting information.

- *SGI Altix ICE 8400 Series System Hardware User's Guide*

Describes the features of the SGI ICE 8400 series systems and provides operating instructions and general troubleshooting information.

- *SGI ICE X System Hardware User Guide*

  Describes the features of the SGI ICE X series systems and provides operating instructions and general troubleshooting information.

Helpful Online Resources

Supportfolio is the SGI support web site, including the SGI Knowledgebase, has links for software supports and updates at: https://support.sgi.com/login.

For a complete list of SGI online resources, see the *SGI Peformance Suite 1.x Start Here.*

Conventions

The following conventions are used throughout this document:

Convention	**Meaning**
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. (Output is shown in nonbold, fixed-space font.)
[ ]	Brackets enclose optional portions of a command or directive line.

...	Ellipses indicate that a preceding element can be repeated.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in any of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system.

- Send mail to the following address:

  SGI
  Technical Publications
  46600 Landing Parkway
  Fremont, CA 94538

SGI values your comments and will respond to them promptly.

Introduction

The *UPC Language Specifications* document defines Unified Parallel C (UPC) as a parallel extension to the C programming language standard that follows the partitioned global address space programming model. It is available at the following location: http://upc.gwu.edu/docs/upc_specs_1.2.pdf.

UPC: Distributed Shared Memory Programming provides information about UPC programming language. For details about this manual and other resources related to UPC, see the preface “About This Manual”.

The UPC common global address space (symmetric multiprocessing (SMP) and nonuniform memory access (NUMA) provides an application with a single shared, partitioned address space, where variables may be directly read and written by any processor, but each variable is physically associated with a single module load `sgi-upc-devel` processor.

This manual documents the SGI implementation of the UPC standard. This chapter covers the following topics:

- "UPC Implementation" on page 1
- "Allinea Distributed Debugging Tool" on page 4
- "Parallel Performance Wizard" on page 4
- "OFED Configuration for UPC" on page 4

UPC Implementation

The SGI implementation of UPC conforms to Version 1.2 standard. Parallel I/O, which is not yet a part of the language, is not supported.

The SGI Unified Parallel C (UPC) compiler man page describes the `sgiupc`(1) command. `sgiupc` is the front-end to the SGI UPC compiler suite. It handles all stages of the UPC compilation process: UPC language preprocessing, UPC-to-C translation, back-end C compilation, and linking with UPC runtime libraries.

To see the sgiupc(1) man page, make sure the sgi-upc-devel module is loaded, as follows:

```
% module load sgi-upc-devel
```

Compiling and Executing a Sample UPC Program

A sample UPC program (hello.c) is, as follows:

```
#include <upc.h>
#include <stdio.h>
int
main ()
{
  printf("Executing on thread %d of %d threads\n", MYTHREAD, THREADS);
}
```

To compile this program and generate the executable hello, use the following command:

```
# sgiupc hello.c -o hello
```

The mpirun(1) command is used for execution. If you want the program to execute using four threads, perform the following command:

```
# mpirun -np 4 hello
```

You can expect output similar to the following:

```
Executing on thread 1 of 4 threads
Executing on thread 3 of 4 threads
Executing on thread 0 of 4 threads
Executing on thread 2 of 4 threads
```

Note: The statements might not appear in the order listed in the output example, above.

For more information on sgiupc(1) and mpirun(1), see the corresponding man pages.

Mixing of UPC Programs with Other Languages

The rules for mixing UPC programs with programs written in other languages are similar to that of mixing a C program compiled with the native compiler used to compile the UPC program (as specified by `UPC_NATIVE_CC`), with the caveat for shared pointers, as follows:

If the main program is compiled using `sgiupc`, the appropriate libraries needed for running UPC programs are linked in. If the main program is not a UPC program compiled with `sgiupc`, the appropriate runtime libraries needed by `sgiupc` have to be explicitly linked in. You can determine this by specifying the `-v` option to the `sgiupc` command used to compile and link an application comprising of a single UPC program.

Shared Pointer Representation and Access

In order to handle large thread counts, as well as large blocking size, the SGI UPC compiler uses a `struct` type to represent a shared pointer. As SGI reserves the right to change this representation at a later time, it would be best to use UPC provided functions to access the individual components if a shared pointer is to be passed to a non-UPC function.

Vectorization of Loops to Reduce Remote Communication Overhead

Consider the following loop:

```
upc_forall (i = 0; i < N; i++; i)
  a[i] = b[i] + c[i];
```

If the array references are all remote, there are 2*N remote loads and N stores performed in this loop.

If the loop does not have any aliasing issues, the number of remote loads can be reduced to 2 and the stores to 1, although each of these would be dealing with N elements at a time. This will cut down the communication overheads to fetch remote data.

If a, b, and c are shared restricted pointers, the compiler is able to figure out that there are no aliasing issues, and it is able to vectorize this loop so that remote block data accesses can be used.

For all other cases, the user can specify a pragma type before the loop, as follows:

```
#pragma sgi_upc vector=on
upc_forall (i = 0; i < N; i++; i)
 a[i] = b[i] + c[i];
```

Note that the upc_forall can contain several statements.

Allinea Distributed Debugging Tool

The Allinea Distributed Debugging Tool (DDT) is an advanced debugging tool available for scalar, multi-threaded, and large-scale parallel applications. DDT 3.1 and later supports sgiupc 1.05 and later. For more information on DDT refer to the ddt command's help option or the following site: http://allinea.com/ddt.

Parallel Performance Wizard

Parallel Performance Wizard (PPW) is a performance analysis tool designed for partitioned global-address-space (PGAS) programs. PPW 2.8 and later supports sgiupc 1.05 and later. For more information on PPW refer to the ppw man page, the ppwhelp command, or the following site: http://ppw.hcs.ufl.edu/.

OFED Configuration for UPC

This section describes how to configure the OpenFabrics Enterprise Distribution (OFED) maximum queue pair (QP) max_qp variable to be able to run SHMEM and UPC on large node count clusters over the OFED fabric running SUSE Linux Enterprise Server 11 Service Pack 1 (SLES 11 SP1), SLES 11 SP2, or Red Hat Enterprise Server 6.2 (RHEL 6.2).

SHMEM and UPC codes need to use the InfiniBand RC protocol for communication between all pairs of processes in the parallel job. This requires a large number of QPs The log(2) of the number of QPs is defined by the log_num_qp and can be configured, as follows:

SLES11 SP1

The mlx4_core module provided by the kernel-default package does **not** have the log_num_qp parameter.

The maximum number of queue pairs (QPs) will default to 2^17 (131072).

The mlx4_core provided by the ofed-kmp-default (from the PTF.693243 update only, not from the standard SLES11 SP1 release), **does** have the log_num_qp parameter.

It can be adjusted by making an addition to the /etc/modprobe.conf.local file, as follows:

```
options mlx4_core log_num_qp=21
```

SLES11 SP2

The mlx4_core module provided by the kernel-default package **does** support the log_num_qp parameter.

The maximum number of QPs will default to 2^17 (131072).

It can be adjusted by making an addition to the /etc/modprobe.conf.local file, as follows:

```
options mlx4_core log_num_qp=21
```

RHEL 6.2

The mlx4_core module provided by the kernel package **does** support the log_num_qp parameter.

The maximum number of QPs will default to 2^17 (131072).

You can adjust this with an entry in the /etc/modprobe.conf file, as follows:

```
options mlx4_core log_num_qp=21
```

Chapter 2

UPC Job Environment

This chapter describes the SGI® UPC run-time environment and covers the following topics:

- "UPC Job Environment" on page 7
- "UPC Quick Start on SGI UV or SGI ICE Systems" on page 7
- "UPC Runtime Library Environment Variables" on page 8

UPC Job Environment

The SGI UPC run-time environment depends on the SGI Message Passing Toolkit (MPT) MPI and SHMEM libraries and the job launch, parallel job control, memory mapping, and synchronization functionality they provide. UPC jobs are launched like MPT MPI or SHMEM jobs, using the `mpirun`(1) or `mpiexec_mpt`(1) commands. UPC thread numbers correspond to `SHMEM PE` numbers and MPI rank numbers for `MPI_COMM_WORLD`.

By default, UPC (MPI) jobs have UPC threads (MPI processes) pinned to successive logical CPUs within the system or cpuset in which the program is running. This is often optimal, but at times there is benefit in specifying a different mapping of UPC threads to logical CPUs. See the MPI job placement information in the `mpi`(1) man page under **Using a CPU List** and `MPI_DSM_CPULIST`, and see the `omplace`(1) man page for more information about placement of parallel MPI/UPC jobs.

UPC Quick Start on SGI UV or SGI ICE Systems

This section describes environment variable settings that may be appropriate for some common UPC program execution situations on SGI® UV™ and SGI® ICE™ systems.

SGI UPC is designed with three options for performing references to non-local portions of shared arrays:

- Processor driven shared memory
- Global reference unit (GRU) driven shared memory

The GRU is a remote direct memory access (RDMA) facility provided by the UV hub application-specific integrated circuit (ASIC).

- InfiniBand fabric driven shared memory access

By default, UPC uses processor-driven references for nearby sockets and GRU-driven references for more distant references. The threshold between "nearby" and "distant" can be tuned with the MPI_SHARED_NEIGHBORHOOD variable, described later in more detail in "UPC Runtime Library Environment Variables" on page 8.

Set the following environment variables:

- Set MPI_GRU_CBS=0

  This makes all GRU resources available to UPC.

- Some SGI UV systems have Intel processors with two hyper-threads per core, while others have a single hyper-thread per core. When dual hyper-threads per core are available, most HPC codes benefit by leaving one hyper-thread per core idle, thereby, giving more cache and functional unit resources to the active hyper-thread that will be assigned to one of the UPC threads. This is easy to do because the upper half of the logical CPUs (by number) are hyper-threads that are paired with the lower half of the logical CPUs. Set GRU_RESOURCE_FACTOR=2 when leaving half of the hyper-threads idle.

- You can experiment with the MPI_SHARED_NEIGHBORHOOD=HOST variable. Some shared array access patterns will be faster using processor-driven references.

- Set GRU_TLB_PRELOAD=100 to get the best GRU-based bandwidth for large block copies.

UPC Runtime Library Environment Variables

The UPC runtime library has a number of environment variables that can affect or tune run-time behavior. They are, as follows:

- UPC_ALLOC_MAX

  This sets the per-thread maximum amount of memory in bytes that can be allocated dynamically by upc_alloc() and the other shared array allocation functions. Note that the SMA_SYMMETRIC_SIZE variable needs to be set to the sum of the value of UPC_ALLOC_MAX plus the amount of space consumed by statically allocated arrays in the UPC program. See the intro_shmem(1) man page for more information about MA_SYMMETRIC_SIZE.

When running UPC programs on InfiniBand clusters, there is particular benefit to setting UPC_ALLOC_MAX to the right size, because physical memory will be pre-allocated in the shared array heap. If the actual memory space utilized by dynamically allocated arrays is less than the pre-allocated amount, excessive physical memory will be consumed.See the intro_shmem(1) man page for more information about SMA_SYMMETRIC_SIZE.

The default is the amount of physical memory per logical CPU on the system.

- UPC_CAUTIOUS_STRICT

  When enabled (nonzero), libupc performs a upc_fence call before all strict accesses, regardless if the previous access was strict or relaxed. When disabled (zero), libupc performs a upc_fence call only if there were one or more relaxed writes since the previous upc_fence.

  The default is disabled.

- UPC_GRU_DOMAIN_SIZE

  This variable controls the use of the GRU, as follows:

  – When non-integer, the GRU is never used.

  – When zero or negative integer, the GRU is always used.

  – When positive power-of-two, the GRU is used except when all threads communicating are within a block of that size.

  – When positive non-power-of-two, rounded down to the next power-of-two.

- UPC_HEAP_CHECK

  When set to 1, causes libupc to check the integrity of the shared memory heap from which shared arrays are allocated.

  The default value is 0.

- UPC_IB_BUFFER_SIZE

  This variable sets the size of the buffer used for InfiniBand fabric copy operations. This per-thread buffer is only allocated and used for remote-to-remote copies over InfiniBand, or any transfers of data to/from InfiniBand where the data cannot be transferred directly.

  The default size is 16 kB. The minimum size is 1 kB.

A number of MPI and SHMEM environment variables described on the MPI(1), SHMEM(1) and gru_resource(3) man pages can be used to tune the execution of UPC programs on SGI UV systems. These man pages should be consulted for a complete list of tunable environment variables. Some of the most helpful variables for UPC programs are, as follows:

- MPI_SHARED_NEIGHBORHOOD

This environment variable has an effect only on SGI UV systems. This variable can be set to HOST to request that UPC shared arrays use processor-driven shared memory transfers instead of GRU transfers. The size of the memory blocks being accessed in a remote part of a shared array and other factors can determine whether processor-driven or GRU-driven transfers will perform better.

The default setting for the MPI_SHARED_NEIGHBORHOOD variable is BLADE, which implies that UPC threads will use processor-driven shared memory for references to shared array blocks that have affinity for the threads associated with sockets on the same UV hub.

- MPI_GRU_CBS and MPI_GRU_DMA_CACHESIZE

  These environment variables have an effect only on SGI UV systems. These variables reserve SGI UV GRU resources for MPI and thereby makes them unavailable for UPC. Setting MPI_GRU_CBS to 0 will have the result of making all GRU resources available to UPC.

- GRU_RESOURCE_FACTOR

  This environment variable has an effect only on SGI UV systems. This environment variable specifies an integer multiplier that increases the amount of per-thread GRU resources that can be used by a UPC program. If UPC programs are placed such that some portion of the logical CPUs (hyper-threads) on each UV hub are left idle, you can specify a corresponding multiplier. For example, if half of the logical CPUs are idle, a setting of GRU_RESOURCE_FACTOR=2 would be recommended. See the gru_resource(3) man page for more details.

Index