GNU UPC

GNU Tools Cauldron 2013

Presenter:

Gary Funck <gary@intrepid.com>

UPC

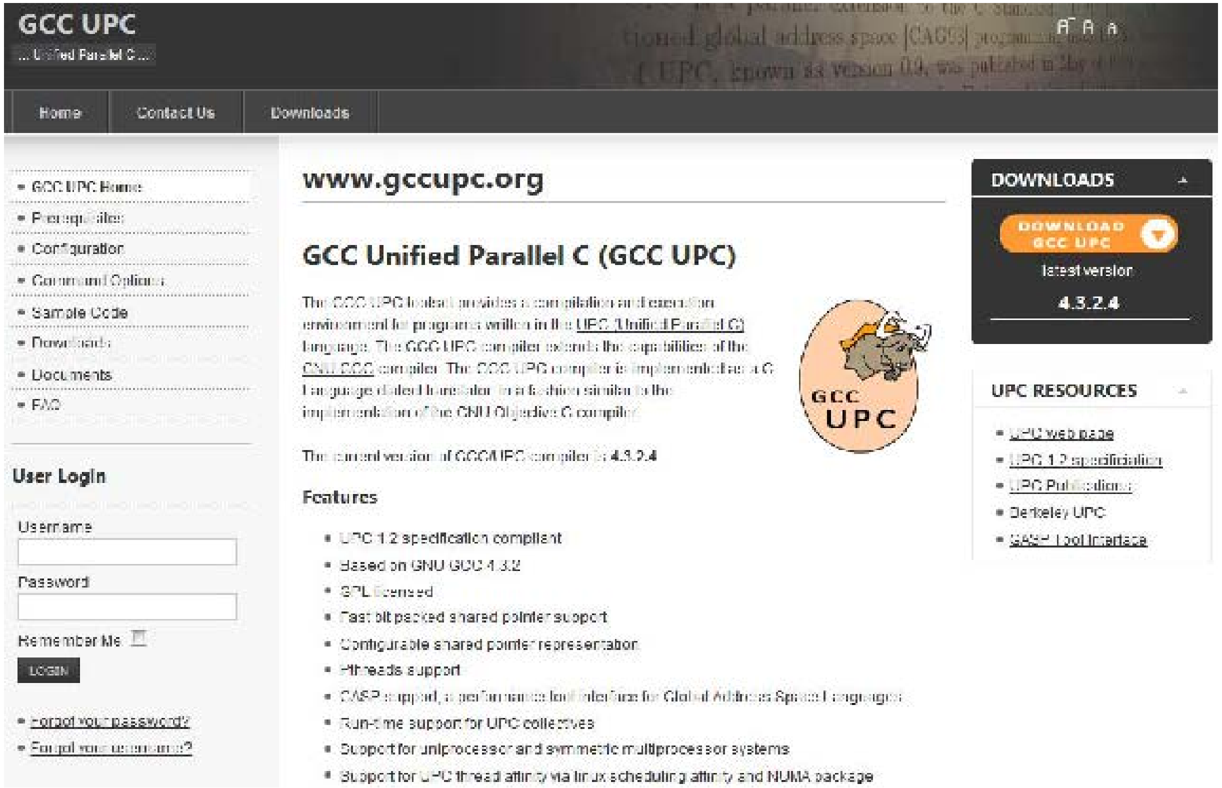
- Unified Parallel C
- Finalized specification (1.0) in 2001
- Unified features found in previous projects (AC, Split-C, PCP)
- Extends ISO C 99 to add explicit parallel language features
- UPC Language Specification v1.2, dated March 2005
- UPC Language Specification v1.3, in progress http://code.google.com/p/upc-specification/

GNU UPC

- First release in year 2000 for SGI IRIX 6.5 was based on GCC 2.9.5.
- Extended earlier 2.7.2 version implemented for Cray T3-E
- Supported architectures: x86_64, x86, PPC/POWER7, IA64, MIPS
- Supported platforms: Linux, Cray XT3/4/5/6, SGI Altix
- GDB support for SMP based runtime only on x86, x86_64 Linux

GNU UPC @ gccupc.org

http://www.gccupc.org/



www.gccupc.org

GCC Unified Parallel C (GCC UPC)

The GCC UPC toolset provides a compilation and execution environment for programs written in the UPC (Unified Parallel C) language. The GCC UPC compiler extends the capabilities of the GNU GCC compiler. The GCC UPC compiler is implemented as a C Language dialect translator, in a fashion similar to the implementation of the GNU Objective C compiler.

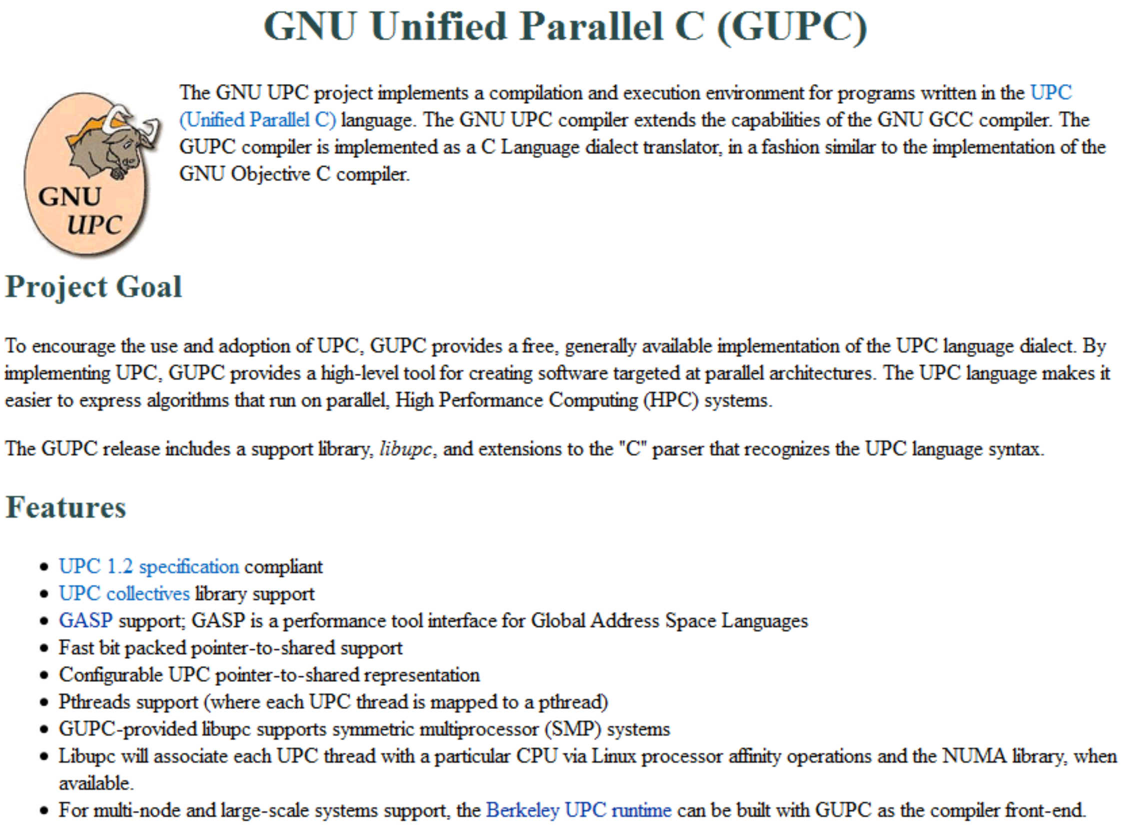
The current version of GCC/UPC compiler is 4.3.2.4

Features

- UPC 1.2 specification compliant
- Based on GNU GCC 4.3.2
- GPL licensed
- Fast bit packed shared pointer support
- Configurable shared pointer representation
- Pthreads support
- GASP support, a performance tool interface for Global Address Space Languages
- Run-time support for UPC collectives
- Support for uniprocessor and symmetric multiprocessor systems
- Support for UPC thread affinity via linux scheduling affinity and NUMA package

GNU UPC @ gnu.org

http://gcc.gnu.org/projects/gupc.html



GNU Unified Parallel C (GUPC)

The GNU UPC project implements a compilation and execution environment for programs written in the UPC (Unified Parallel C) language. The GNU UPC compiler extends the capabilities of the GNU GCC compiler. The GUPC compiler is implemented as a C Language dialect translator, in a fashion similar to the implementation of the GNU Objective C compiler.

Project Goal

To encourage the use and adoption of UPC, GUPC provides a free, generally available implementation of the UPC language dialect. By implementing UPC, GUPC provides a high-level tool for creating software targeted at parallel architectures. The UPC language makes it easier to express algorithms that run on parallel, High Performance Computing (HPC) systems.

The GUPC release includes a support library, *libupc*, and extensions to the "C" parser that recognizes the UPC language syntax.

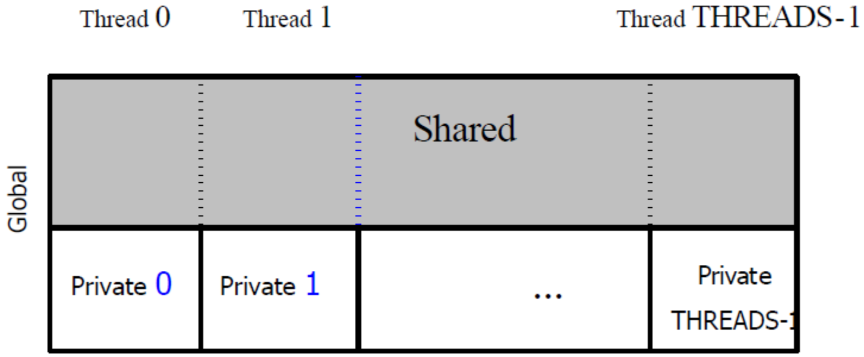
Features

- UPC 1.2 specification compliant
- UPC collectives library support
- GASP support; GASP is a performance tool interface for Global Address Space Languages
- Fast bit packed pointer-to-shared support
- Configurable UPC pointer-to-shared representation
- Pthreads support (where each UPC thread is mapped to a pthread)
- GUPC-provided libupc supports symmetric multiprocessor (SMP) systems
- Libupc will associate each UPC thread with a particular CPU via Linux processor affinity operations and the NUMA library, when available.
- For multi-node and large-scale systems support, the Berkeley UPC runtime can be built with GUPC as the compiler front-end.

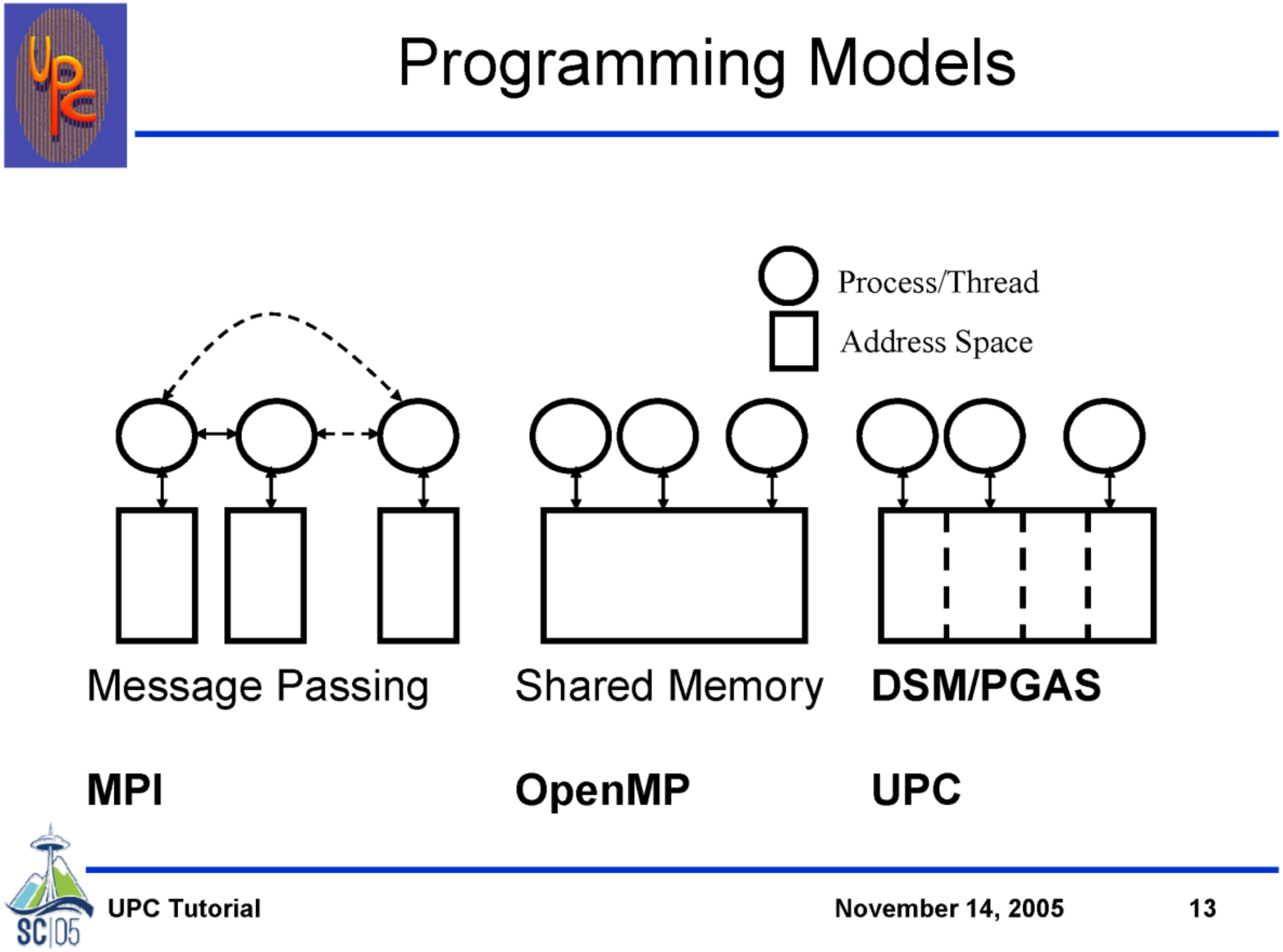
UPC Language

UPC Global Shared Memory

- A collection of threads operating in a single global address space
- Address space is logically partitioned among threads
- Each thread has affinity with a portion of the globally shared address space
- Each UPC thread is usually a separate Unix process



UPC Programing Model



Programming Models

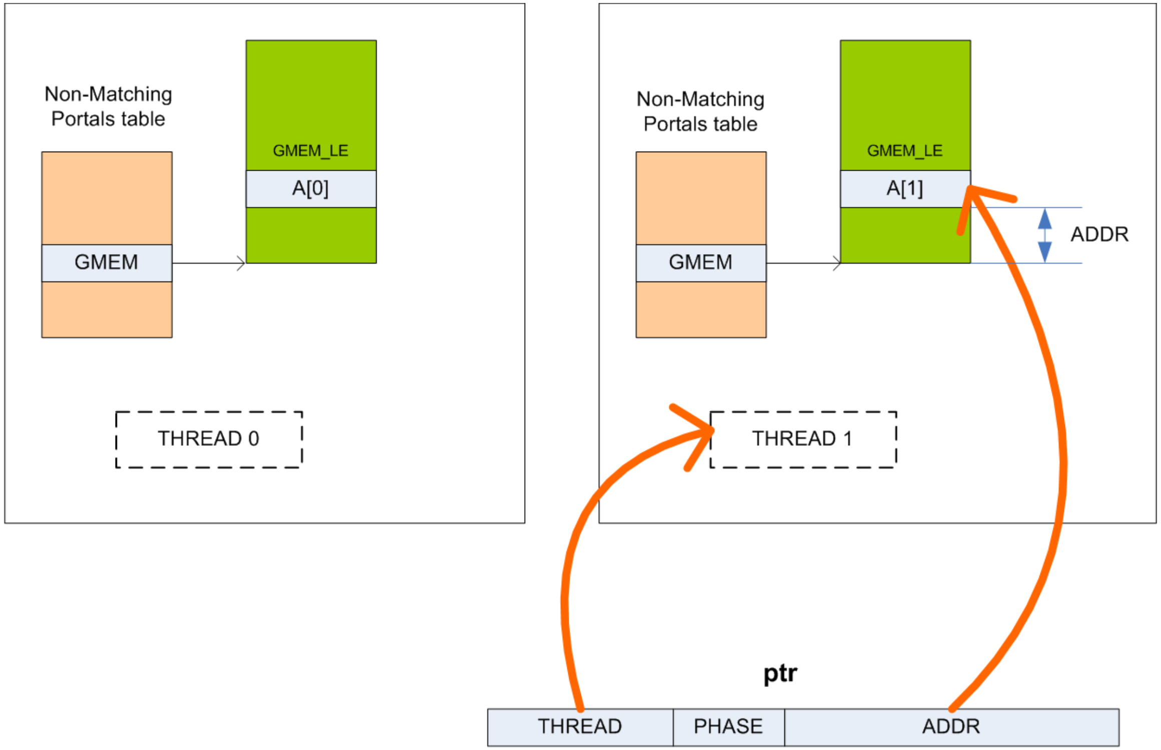
○ Process/Thread

□ Address Space

Message Passing	Shared Memory	**DSM/PGAS**
MPI	**OpenMP**	**UPC**

UPC Tutorial **November 14, 2005** **13**

Shared Memory Access via UPC Pointer-to-Shared



```
shared int A[THREADS];
shared int *ptr;
[...]
ptr = &A[1];
```

UPC Shared Declarations

- Declared as *“shared”* qualified

```
shared int i;
```

Integer allocated on Thread 0.

```
shared int A[THREADS];
```

Shared array. Integer allocated on each thread.

```
shared [2] int B[THREADS];
```

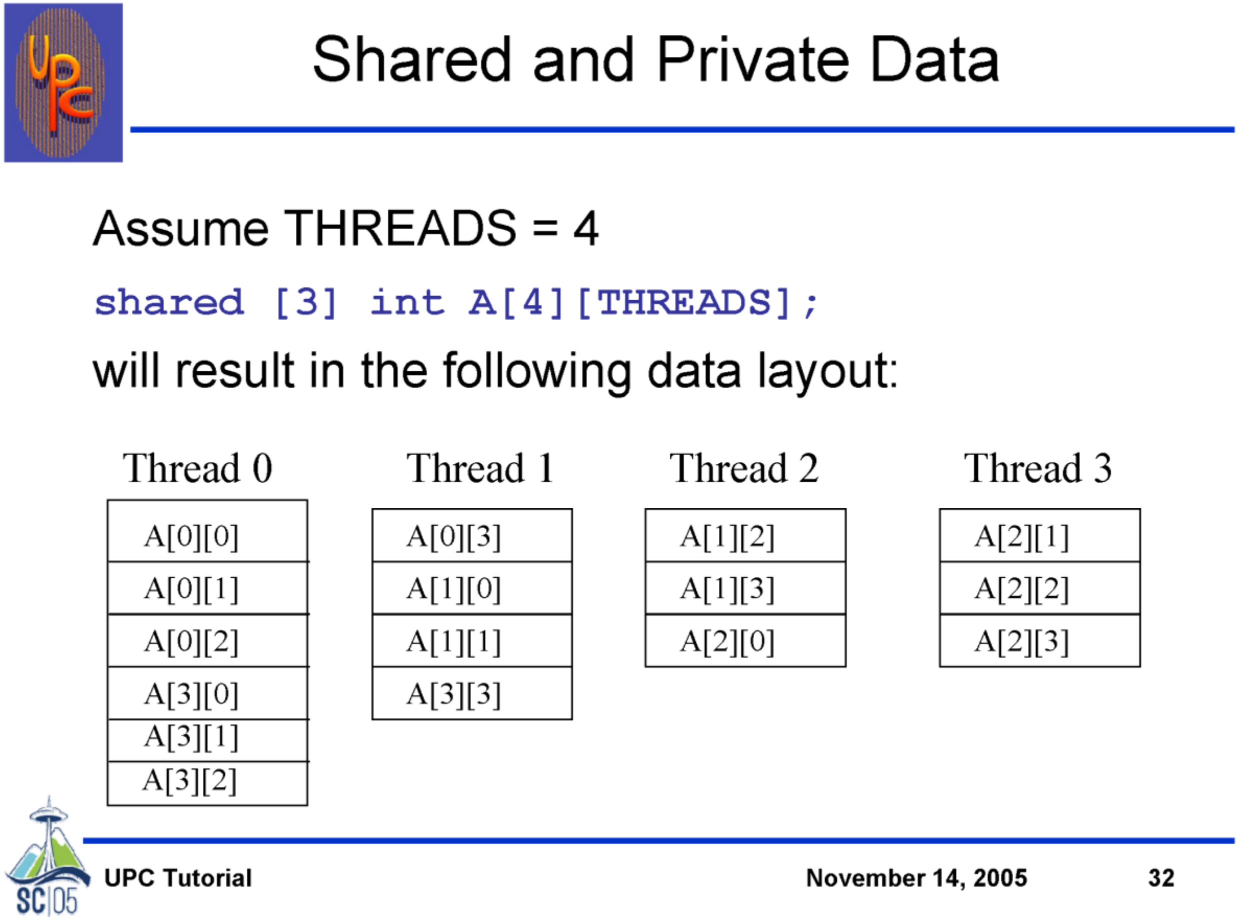
Shared array with a layout qualifier.

B[0] and B[1] allocated on Thread 0.

```
shared int *p;
```

Pointer to a shared object. Storage for pointer is in local/private address space.

UPC Shared Data Layout



Shared and Private Data

Assume THREADS = 4

```
shared [3] int A[4][THREADS];
```

will result in the following data layout:

Thread 0	Thread 1	Thread 2	Thread 3
A[0][0]	A[0][3]	A[1][2]	A[2][1]
A[0][1]	A[1][0]	A[1][3]	A[2][2]
A[0][2]	A[1][1]	A[2][0]	A[2][3]
A[3][0]	A[3][3]		
A[3][1]			
A[3][2]			

UPC Tutorial November 14, 2005 32

UPC Memory Model

- Relaxed memory accesses
  ```
  relaxed shared int i;
  ```
- Strict memory accesses
  ```
  strict shared int i;
  ```
- Default memory model: relaxed
- Change memory model with pragmas
  - `#pragma upc strict`
  - `#pragma upc relaxed`

UPC Memory Consistency

- Relaxed accesses may appear to be arbitrarily reordered relative to program order unless:
  - access is to the same location and at least one of them is a write
- Strict accesses must
  - appear in program order with respect to other strict accesses
  - appear after all relaxed accesses complete
  - complete before any relaxed access

UPC Example: Parallel Sum

```
shared int A[100*THREADS];
shared int tsum[THREADS];
int i, sum = 0;
for (i = MYTHREAD; i < 100*THREADS; i += THREADS)
  A[i] = i + 1;
upc_barrier;
for (i = MYTHREAD; i < 100*THREADS; i += THREADS)
  sum += A[i];
tsum[MYTHREAD] = sum;
upc_barrier;
if (MYTHREAD == 0)
  {
    /* sum up each thread’s partial sum */
    for (i = 1; i < THREADS; ++i)
      sum += tsum[i];
    printf ("Sum = %d\n", sum);
  }
```

UPC Pointer-to-shared Arithmetic

```
shared [B] int *p1, *p2;
int i;
p2 = p1 + i;
```

Equivalent “C” logic

```
a = (i < 0) ? 1 : 0;
ph_tmp = p1.phase - a * (B - 1);
th_tmp = p1.thread - a * (THREADS - 1);
p2.phase = (ph_tmp + i) % B + a * (B - 1);
p2.thread = (th_tmp + (ph_tmp + i) / B) % THREADS +
                      a * (THREADS - 1);
block_incr = (th_tmp + (ph_tmp + i) / B) / THREADS;
elem_incr = (p2.phase – p1.phase) + B * block_incr;
p2.addr = p1.addr + elem_incr * upc_elemsizeof(*p1);
```

UPC Cast to Local Address

```
int *P1;
shared int *S1;
P1 = (int *) S1; /* allowed if upc_threadof(S1) == MYTHREAD */
```

Bytes with affinity to a given thread containing shared objects can be accessed by either pointers-to-shared or pointers-to-local of that thread.

UPC Synchornization Statements

upc_notify *expression*$_{opt}$	Notify on exiting synchronization phase
upc_wait *expression*$_{opt}$	End of the synchronization phase
upc_barrier *expression*$_{opt}$	Equivalent to **upc_notify** *expression*$_{opt}$ **upc_wait** *expression*$_{opt}$
upc_fence	All shared accesses must complete

- Collective operations
- Each thread executes an alternating sequence of upc_notify and upc_wait statements

UPC Iteration Statement

upc_forall (*expression*$_{opt}$**;** *expression*$_{opt}$**;** *expression*$_{opt}$**;** *affinity*$_{opt}$**)** *statement*

- Collective operation
- The affinity field specifies the executions of the loop body which are to be performed by a thread

GNU UPC

GNU UPC

- UPC 1.2 specification compliant
- UPC 1.3 specification will be finalized soon. Most version 1.3 support is implemented.
- Current released version is based on GCC 4.8
- GNU UPC is available in the GUPC branch of the GCC svn repository
- The current differences between the GUPC branch and the main GCC trunk are detailed at http://www.gccupc.org/gupc-changes

GNU UPC Runtime Library

- The UPC runtime makes extensive use of inlining to improve performance
- Built-in runtime support for symmetric multiprocessor (SMP) systems (*libgupc/smp*)
- Built-in runtime support for multi-node systems with Infiniband interconnects supporting the Portals4 library (*libgupc/portals4*). Requires Portals4 reference implementation (http://code.google.com/p/portals4/).
- Additional multi-node and high speed interconnect support provided by linking with the Berkeley UPC run-time

GNU UPC
PTS Representation

Thread	Phase	Address

Representation	**Size (in bits)**	**Thread**	**Phase**	**Address**
Packed	64	10	20	34
Struct	128	32	32	64

GNU UPC
Shared Access API

- UPC runtime calls are generated by GNU UPC to implement access to UPC shared memory locations
- Type-specific access routines
  - Relaxed Get:

```
u_intSI_t __getsi2 (upcr_shared_ptr_t src);
```

  - Relaxed Put:

```
void __putsi2 (upcr_shared_ptr_t dest, u_intSI_t v);
```

  - Strict Get:

```
u_intSI_t __getssi2 (upcr_shared_ptr_t src);
```

  - Strict Put:

```
void __putssi2 (upcr_shared_ptr_t dest, u_intSI_t v);
```

GNU UPC Runtime Calls - Put

```
shared int A[100*THREADS];
int i, sum = 0;
/* each thread fills in its contribution. */
for (i = MYTHREAD; i < 100*THREADS; i += THREADS)
  A[i] = i + 1;
upc_barrier;
```

```
shared int A[100*THREADS];
int i, sum = 0;
/* each thread fills in its contribution. */
for (i = MYTHREAD; i < 100*THREADS; i += THREADS)
 {
   __putsi2(&A[i], i + 1);
 }
__upc_barrier(ANON_BARRIER_ID);
```

GNU UPC Runtime Calls - Get

```
if (MYTHREAD == 0)
    {
      /* serialized sum on thread 0. Here, thread 0
         accesses the data on all threads. */
      for (i = 0; i < 100*THREADS; ++i)
        sum += A[i];
      printf ("Sum = %d\n", sum);
    }
```

```
if (MYTHREAD == 0)
    {
      for (i = 0; i < 100*THREADS; ++i)
        sum += __getsi2(&A[i]);
      printf ("Sum = %d\n", sum);
    }
```

GNU UPC
Control Flow

GCC Front-End

- **Parses UPC *shared*, *strict*, *relaxed*, and *layout* (blocksize) qualifiers**
- **Parses UPC statements (e. g. *upc_forall*, *upc_barrier)***
- **Implements UPC semantic checks**

GNU UPC Lowering Pass (*upc_genericize.c*)

- **Rewrites UPC-specific tree nodes into SIMPLE. For example, translates UPC shared references to *get* and *put* runtime calls.**
- **Generates initialization code when shared address expressions are used in static initializers.**

GNU UPC
GCC Language Hooks

LANG_HOOKS_NAME	"GNU UPC"
LANG_HOOKS_EXPAND_CONSTANT	Convert constant expressions involving UPC pointers-to-shared into equivalent “C” representation
LANG_HOOKS_GET_ALIAS_SET	Handle aliasing for references to UPC *shared* objects
LANG_HOOKS_GENERICIZE	**New: UPC lowering pass**
LANG_HOOKS_HANDLE_OPTION	Handle UPC-specific options
LANG_HOOKS_INIT	Initialize UPC-specific processing
LANG_HOOKS_FINISH	Finalize UPC-specific processing
LANG_HOOKS_INITIALIZE_DIAGNOSTICS	Initialize UPC-specific diagnostics
LANG_HOOKS_INIT_OPTIONS	Initialize UPC-specific options processing
LANG_HOOKS_POST_OPTIONS	Finalize UPC-specific options processing
LANG_HOOKS_TYPES_COMPATIBLE_P	Check compatibility for UPC *shared* objects and references to those objects
LANG_HOOKS_INIT_TS	Register UPC-specific “tree contains struct” tree nodes

GNU UPC
Key gcc/*upc*/ Source Files

config-lang.in	UPC-specific *configure* processing
gupcspec.c	UPC command line driver
gupc.texi	UPC documentation
Make-lang.in	UPC-specific *make* rules
upc-act.c	UPC-related actions: supports both the front-end and lowering pass
upc-gasp.c	UPC-specific performance-related instrumentation
upc-genericize.c	UPC lowering pass (rewrites UPC constructs into “C” constructs)
upc-lang.c	UPC-specific language hooks
upc-pts-packed.c	UPC “packed” pointer-to-shared representation manipulation
upc-pts-struct.c	UPC “struct” pointer-to-shared representation manipulation
upc-tree.def	UPC-specific tree node definitions

GNU UPC
GCC Tree Node Extensions

```
 union {
    /* The bits in the following
structure should only be used with
       accessor macros that
constrain inputs with tree checking.
*/
    struct {
      [...]
      unsigned unsigned_flag : 1;
      unsigned packed_flag : 1;
      unsigned user_align : 1;
      unsigned nameless_flag : 1;
      unsigned upc_shared_flag : 1;
      unsigned upc_strict_flag : 1;
      unsigned upc_relaxed_flag : 1;
      [...]
      unsigned address_space : 8;
    } bits;
```

```
/* Non-zero if the UPC blocking factor is 0.
*/
#define TYPE_HAS_BLOCK_FACTOR_0(TYPE)
TYPE_LANG_FLAG_4 (TYPE)

/* Non-zero if the UPC blocking factor is
greater than 1.
   In this case, the blocking factor value
is stored in a hash table.  */
#define TYPE_HAS_BLOCK_FACTOR_X(TYPE)
TYPE_LANG_FLAG_5 (TYPE)

/* Non-zero if the UPC blocking factor is
not equal to 1 (the default).  */
#define TYPE_HAS_BLOCK_FACTOR(TYPE) \
  (TYPE_SHARED(TYPE) \
   && (TYPE_HAS_BLOCK_FACTOR_0 (TYPE) \
       || TYPE_HAS_BLOCK_FACTOR_X (TYPE)))

extern void upc_block_factor_insert (tree,
tree);
extern tree upc_block_factor_lookup
(const_tree);
```

GNU UPC
Technical Issues

- Can GNU UPC be incorporated into GCC without the need to build it as a separate language front end?
- Can SIMPLE (and GIMPLE) be extended to include UPC pointers-to-shared (ie, “fat pointers” used by GNU UPC)?
- Can/should GNU UPC be changed such that it does not depend upon separate linker sections?