Biblioteca de Operaciones Colectivas para el Lenguaje de Programación Paralela UPC

José M. Andión, Guillermo L. Taboada, Juan Touriño y Ramón Doallo

computer architecture group

18 de septiembre de 2009

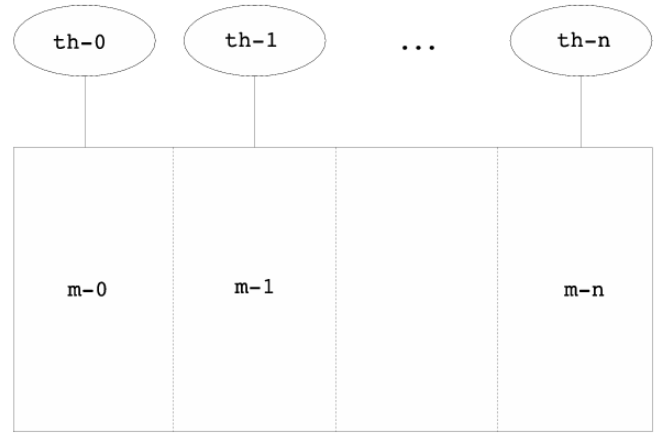
XX Jornadas de Paralelismo – A Coruña

Índice

Programación Paralela

Lenguaje / Paradigma	Fortran	C	Java
Biblioteca de Paso de Mensajes	MPI	MPI	MPJ
Directivas de Memoria Compartida	OpenMP	OpenMP	Threads
PGAS	Co-Array Fortran	**UPC** 	Titanium
HPCS	Fortress	Chapel	X10

UPC (1)

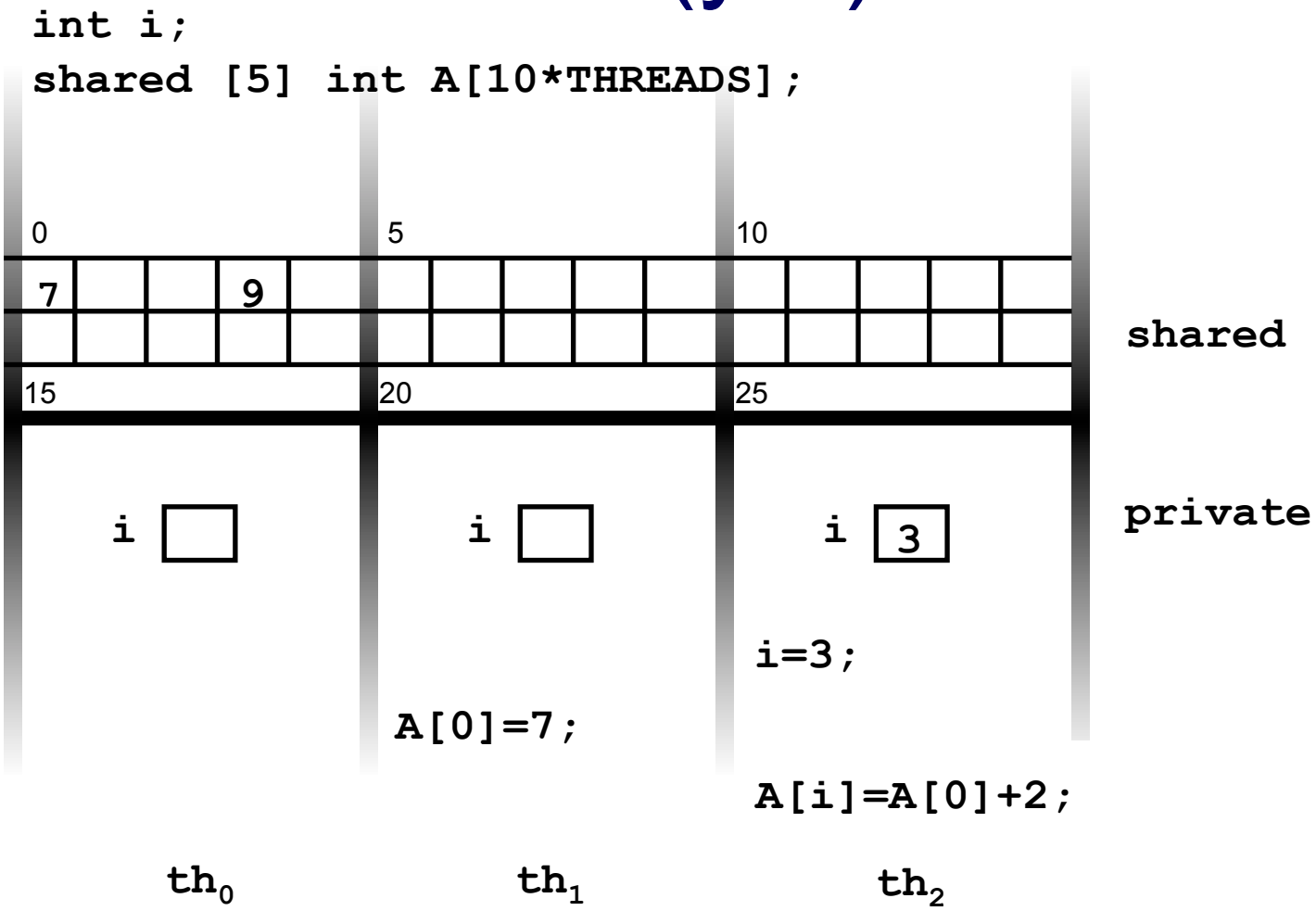


- Unified Parallel C
  - Extensión de C
  - Aproximación PGAS
    - Espacio compartido particionado
    - Paradigma SPMD (Single ProgramMultiple Data)

```
#include <upc.h>
#include <stdio.h>
int main() {
  printf("Hello World from THREAD %d (of %d THREADS)\n",
 MYTHREAD, THREADS);
}
```

```
$ upcrun -n 4 helloworld.out
Hello World from THREAD 0 (of 4 THREADS)
Hello World from THREAD 3 (of 4 THREADS)
Hello World from THREAD 1 (of 4 THREADS)
Hello World from THREAD 2 (of 4 THREADS)
```

UPC (y 2)

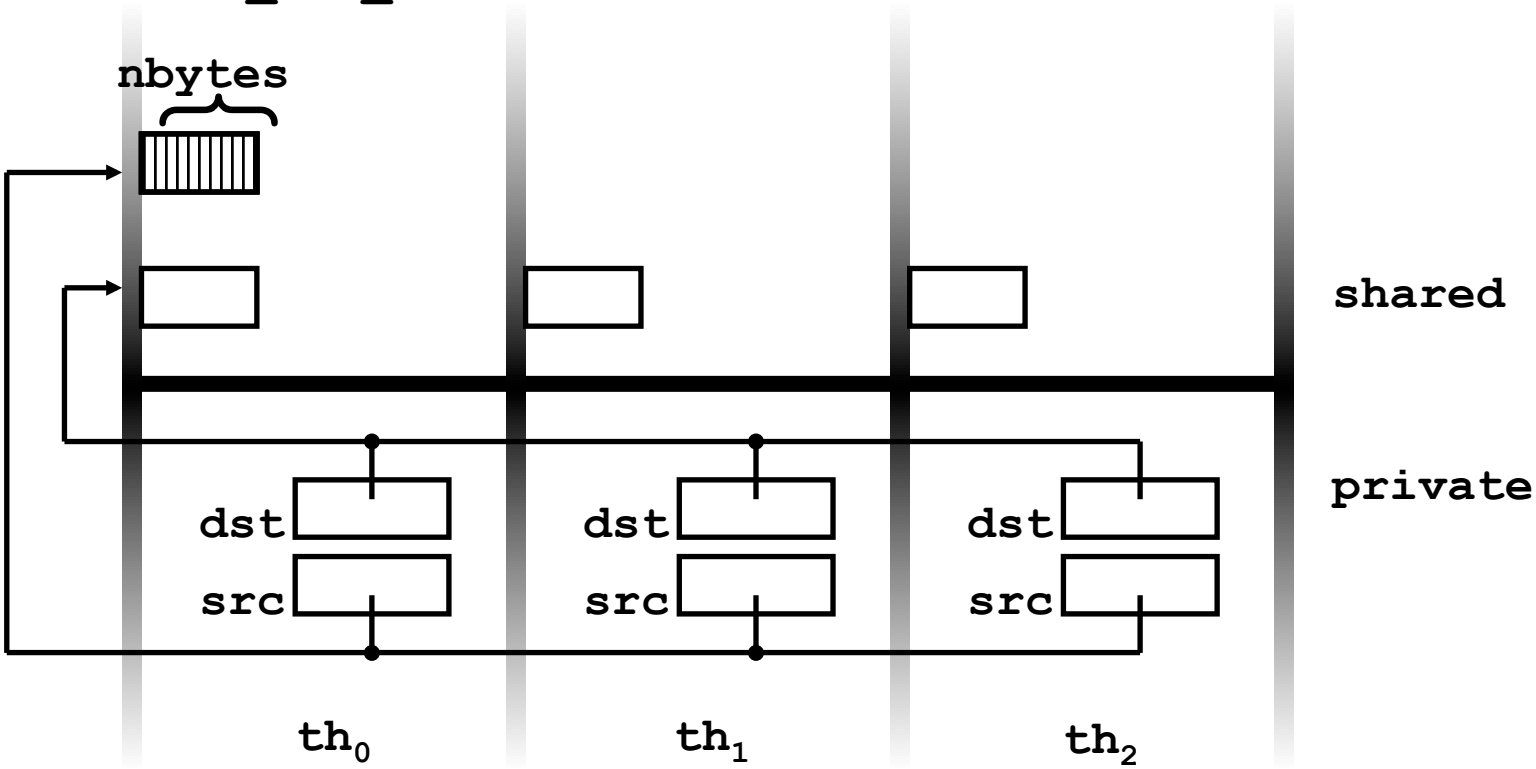

int i;
shared [5] int A[10*THREADS];
0
5
10
7
9
15
20
25
shared
i
i
i
3
private
i=3;
A[0]=7;
A[i]=A[0]+2;
th0
th1
th2

Operaciones Colectivas

- En programación paralela, los procesos han de trabajar coordinadamente para resolver un determinado problema
- Las operaciones que simultáneamente involucran a un grupo de procesos se denominan operaciones colectivas
- Las más comunes integran la biblioteca de operaciones colectivas (**`upc_collective.h`**)
- Dos grandes grupos:
  - Operaciones de relocalización de datos (broadcast, scatter, gather…)
  - Operaciones de computación o consolidación de datos (reduce…)

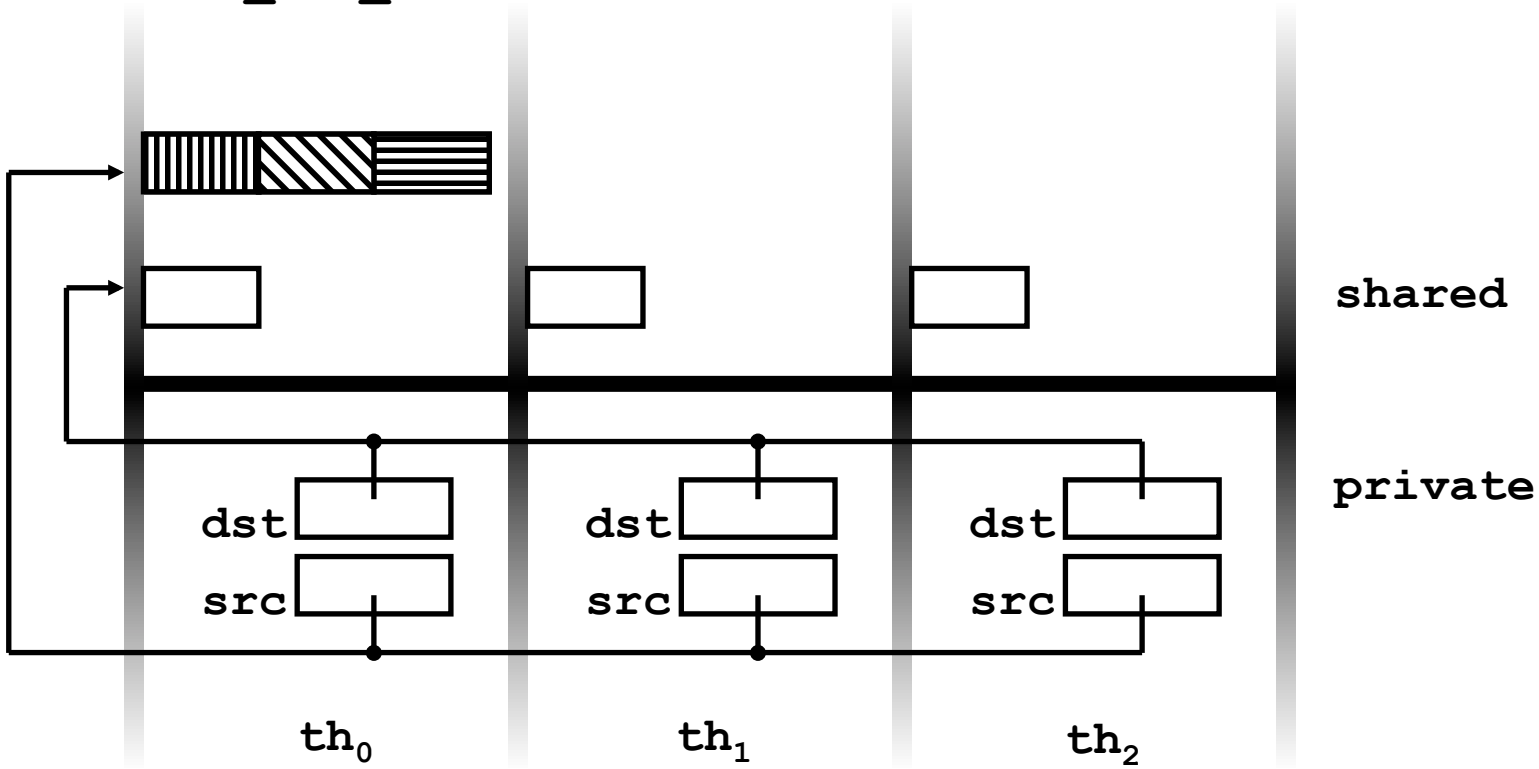
Colectivas en UPC (1)

```
shared [] char src[nbytes];
shared [nbytes] char dst[nbytes * THREADS];
void upc_all_broadcast(dst, src, nbytes, …);
```



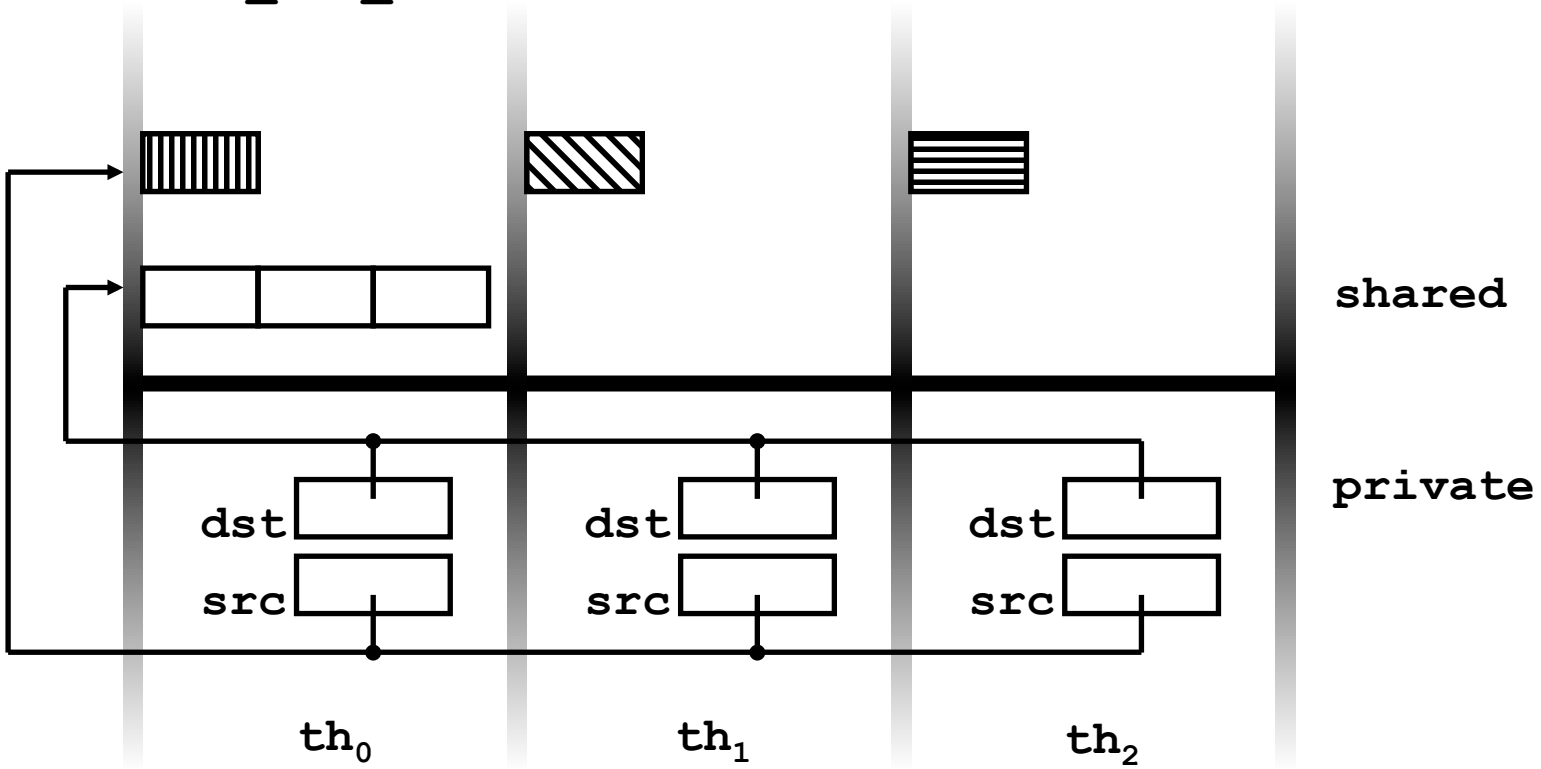
Colectivas en UPC (2)

```
shared [] char src[nbytes * THREADS];
shared [nbytes] char dst[nbytes * THREADS];
void upc_all_scatter(dst, src, nbytes, …);
```



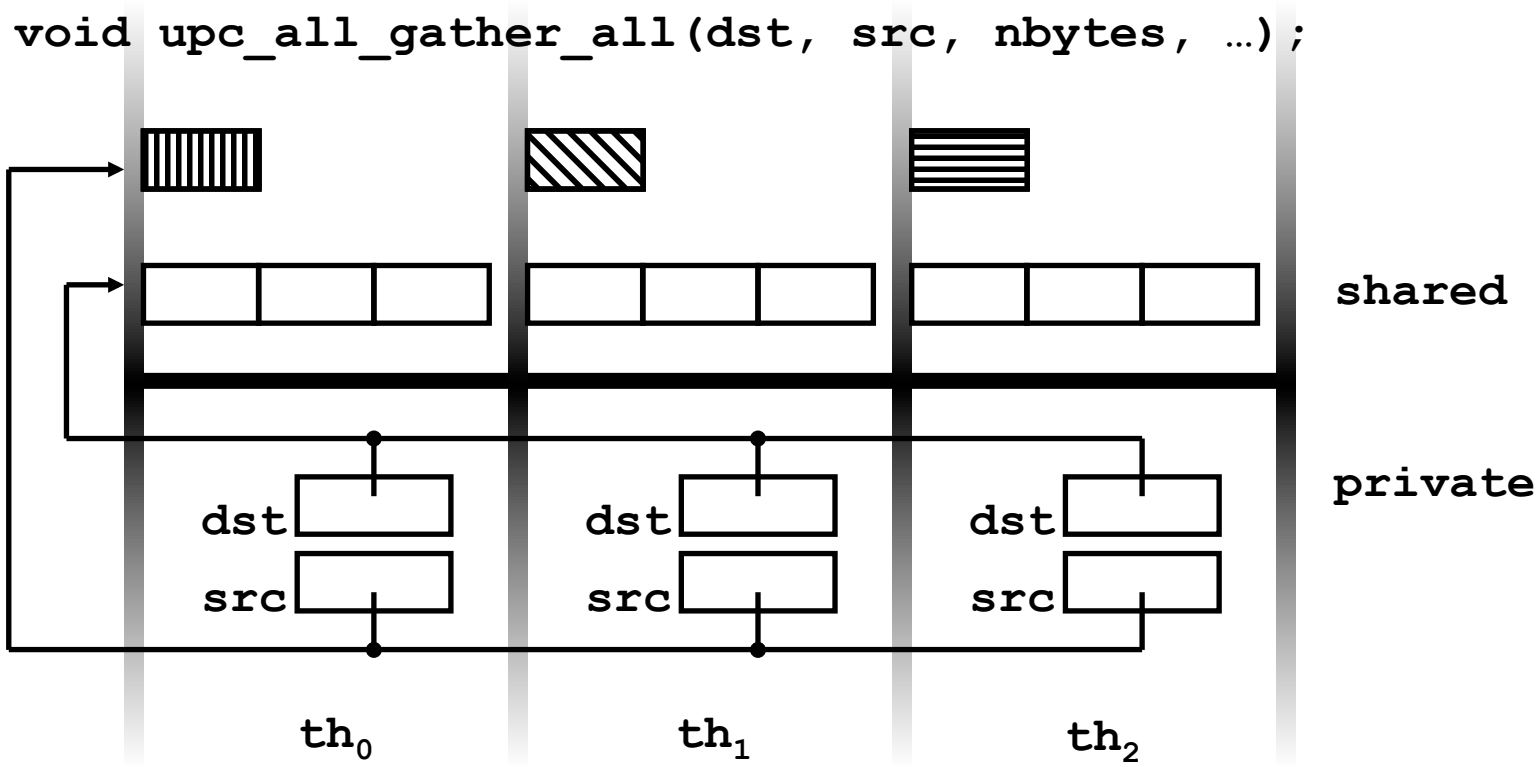
Colectivas en UPC (3)

```
shared [nbytes] char src[nbytes * THREADS];
shared [] char dst[nbytes * THREADS];
void upc_all_gather(dst, src, nbytes, …);
```



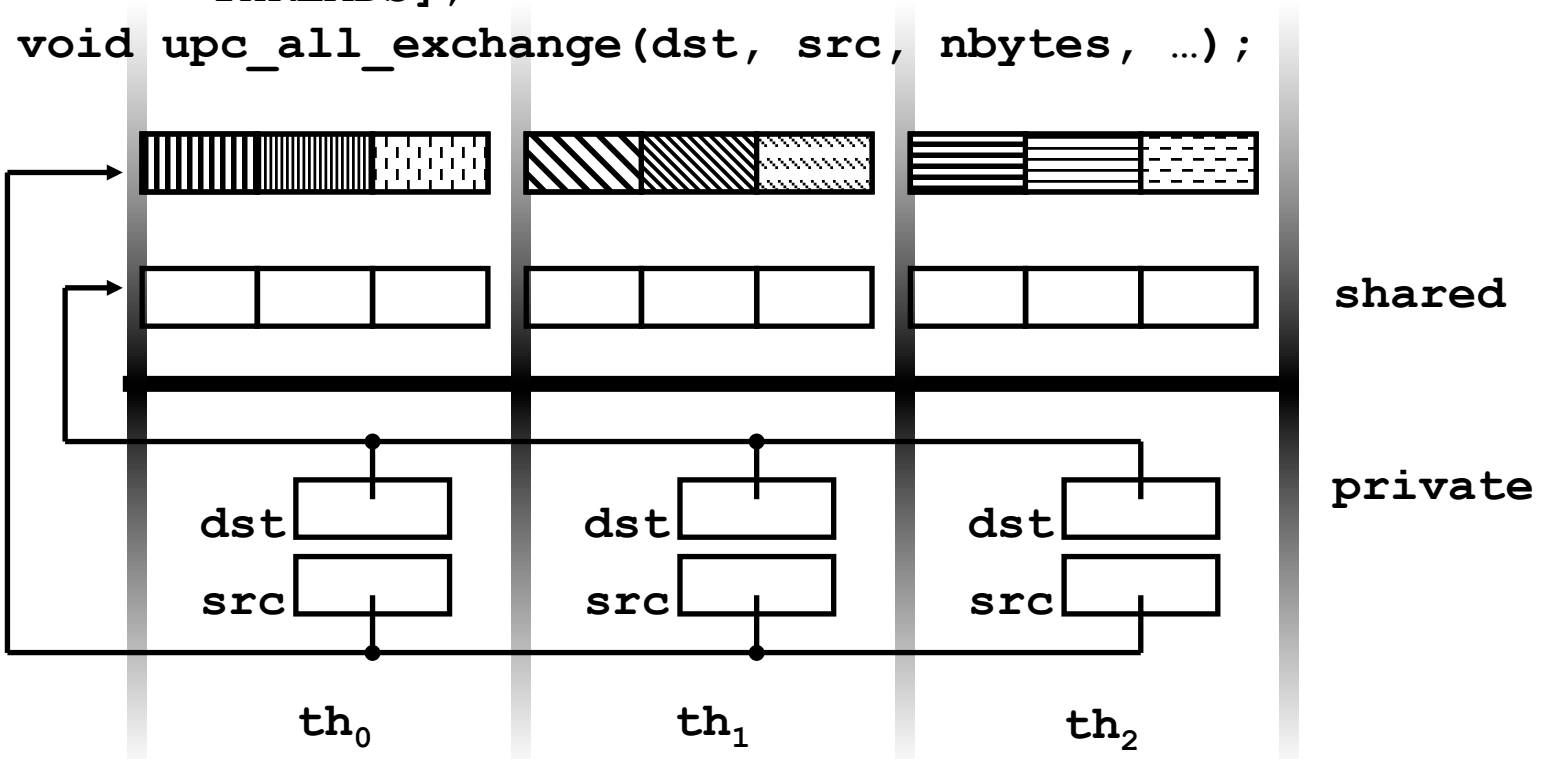
Colectivas en UPC (4)

```
shared [nbytes] char src[nbytes * THREADS];
shared [nbytes * THREADS] char dst[nbytes * THREADS *
       THREADS];
void upc_all_gather_all(dst, src, nbytes, …);
```



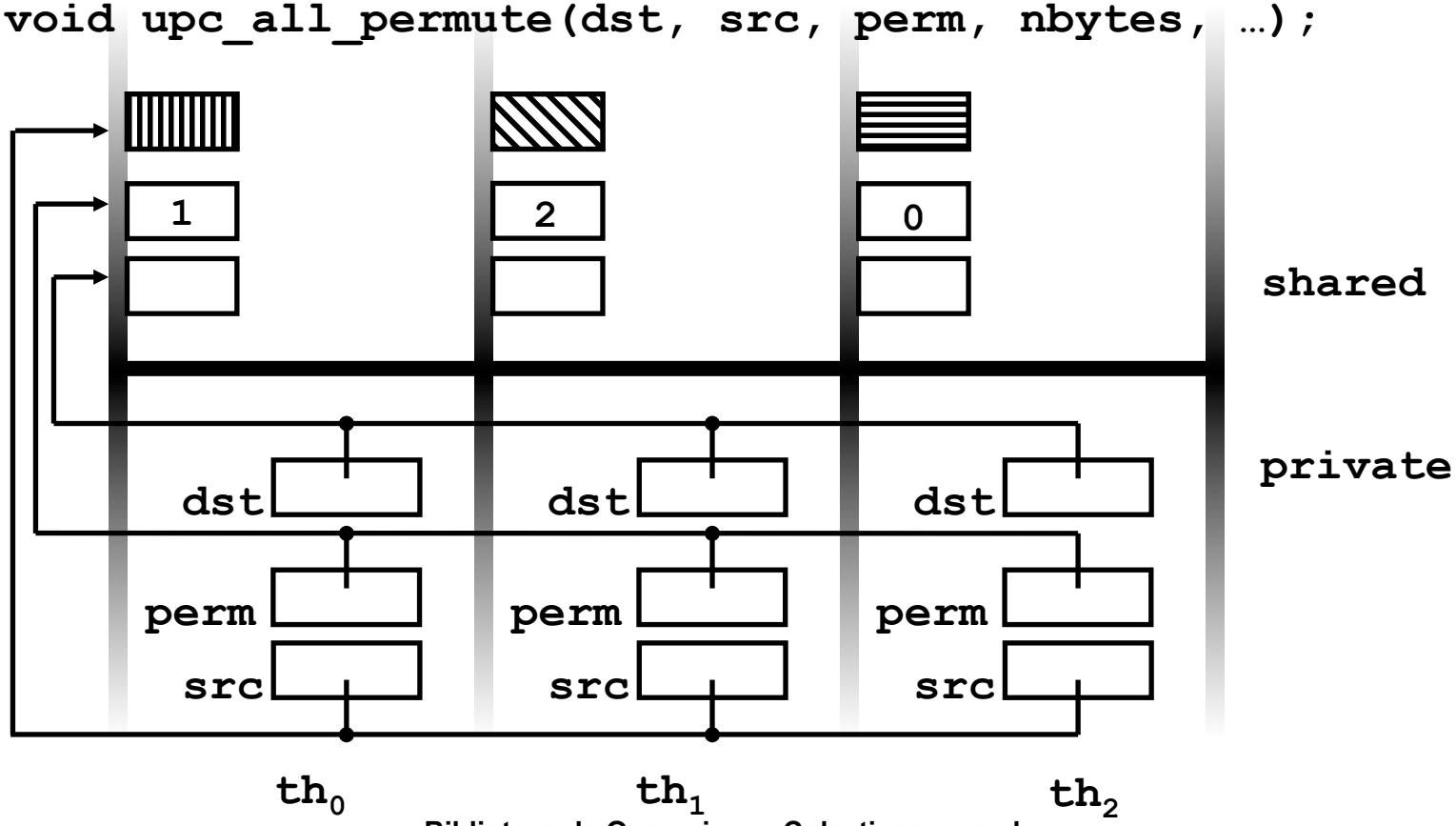
Colectivas en UPC (5)

```
shared [nbytes * THREADS] char src[nbytes * THREADS *
       THREADS];
shared [nbytes * THREADS] char dst[nbytes * THREADS *
       THREADS];
void upc_all_exchange(dst, src, nbytes, …);
```



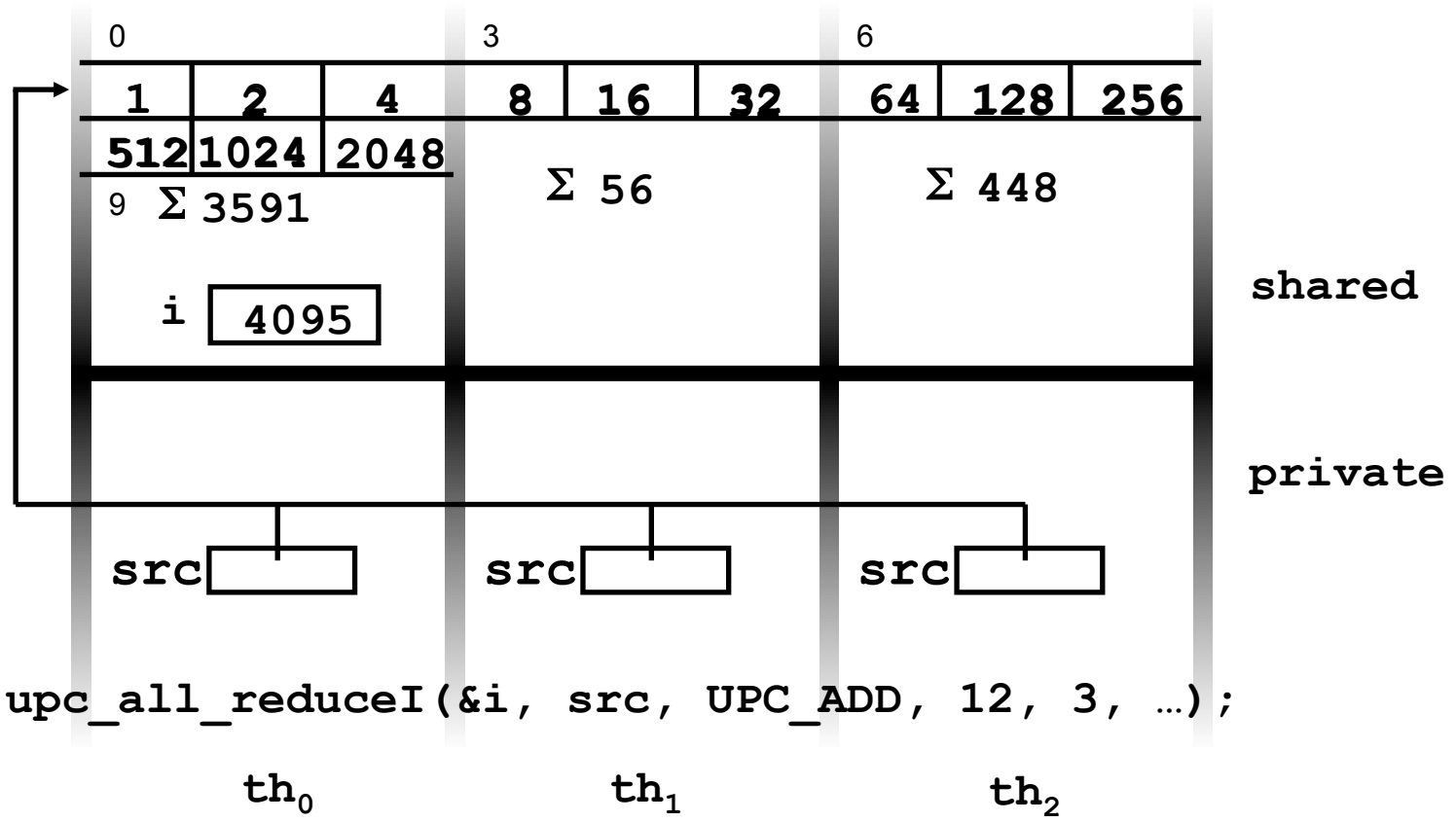
Colectivas en UPC (6)

```
shared [nbytes] char src[nbytes * THREADS];
shared [nbytes] char dst[nbytes * THREADS];
void upc_all_permute(dst, src, perm, nbytes, …);
```



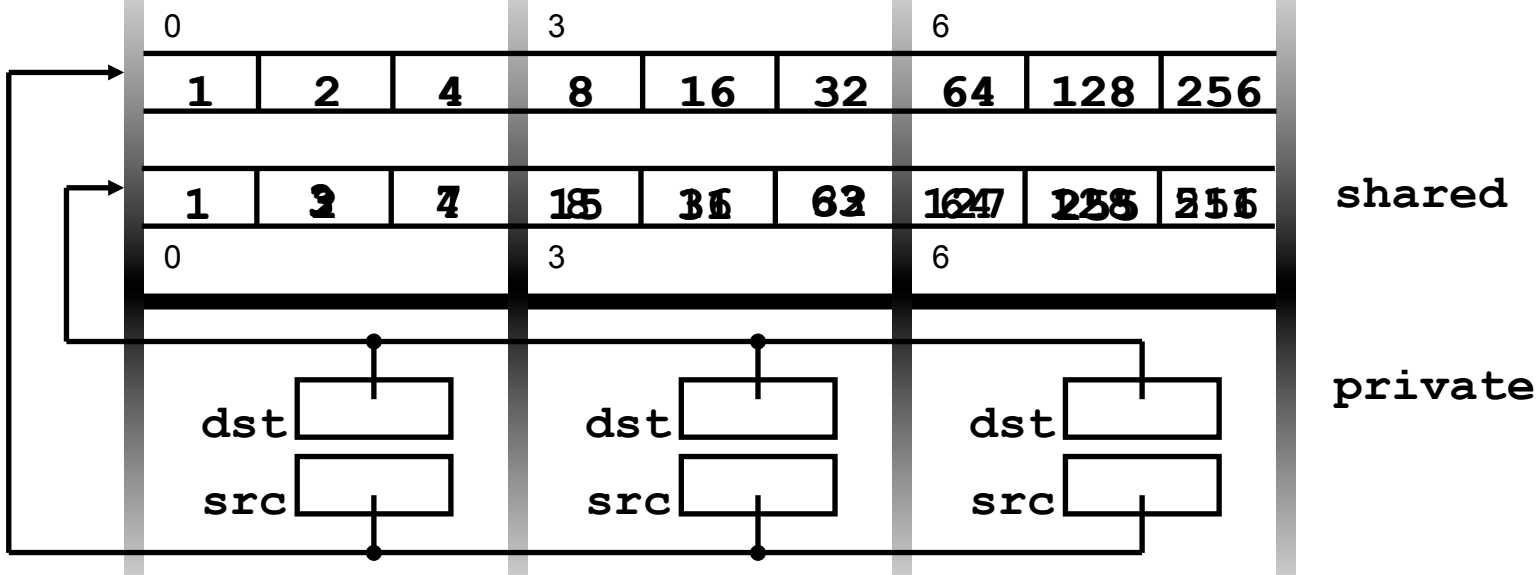
Colectivas en UPC (7)

```
shared int i;
shared [3] int src[12];
```



Colectivas en UPC (y 8)

```
shared [3] int src[9];
shared [3] int dst[9];
```




```
upc_all_prefix_reduceI(dst, src, UPC_ADD, 9, 3, …);
```

th_0 th_1 th_2

Motivación del Trabajo (1)

- Marzo 2008: HP instala en el CESGA el supercomputador FinisTerrae
  - Convenio “ImprovingUPC Usability and Performance in Constellation Systems: Implementation / Extension of UPC Libraries”
  - Hito “Design and Developement of New Collective Operations in UPC”
- HP es un miembro destacado del UPC Consortium, que ha dado importancia a las operaciones colectivas

Motivación del Trabajo (2)

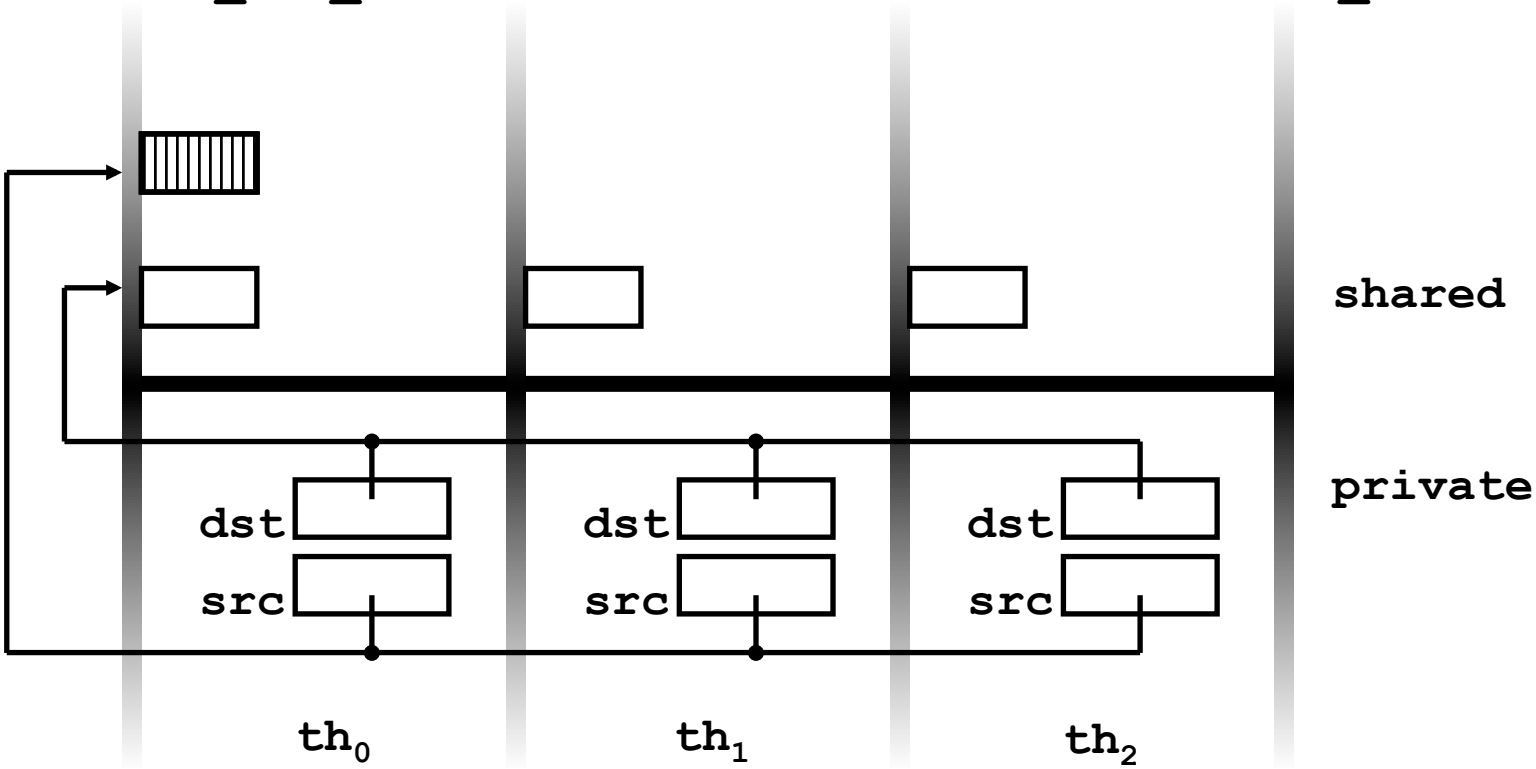
Limitaciones	Solución	Disponibilidad	Trabajo
No optimizadas para sistemas multi-core			
Funcionalidad reducida	Parcialmente	Parcialmente	
Arrays en memoria compartida	Value-Based	Parcialmente	
Mismo tamaño en todos los threads Datos al principio de cada bloque	Vector-Variant	Parcialmente	

Optimización del Rendimiento de Operaciones Colectivas

- Se han implementado 3 algoritmos por operación colectiva
- Seleccionables mediante **`upc_flag_t`**
  - **`UPC_PULL`**
  - **`UPC_PUSH`**
  - **`UPC_MULTICORE`**
- Elegir el de mayor rendimiento

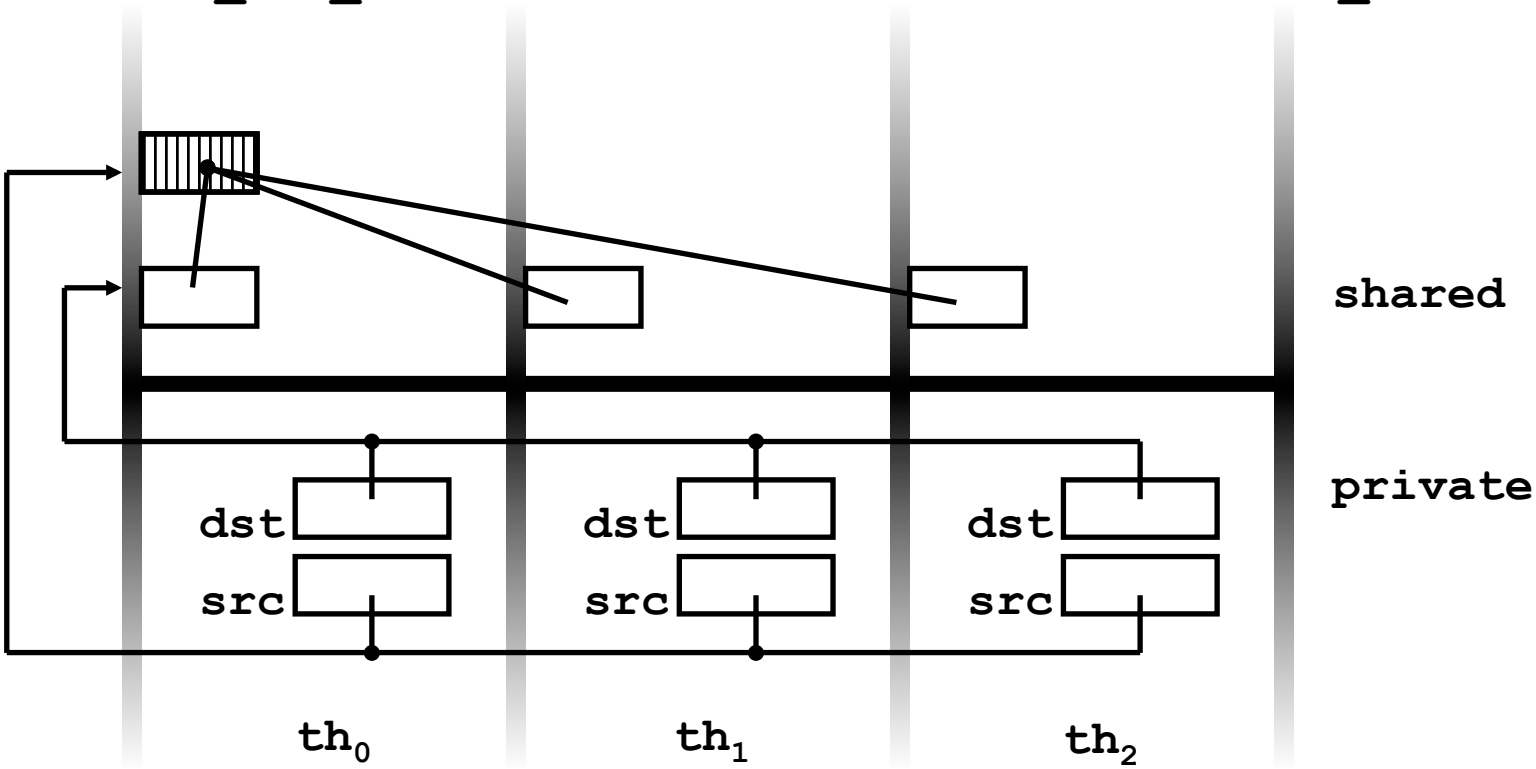
Algoritmo Push

```
shared [] char src[nbytes];
shared [nbytes] char dst[nbytes * THREADS];
void xupc_all_broadcast(dst, src, nbytes, …, UPC_PUSH);
```



Algoritmo Pull

```
shared [] char src[nbytes];
shared [nbytes] char dst[nbytes * THREADS];
void xupc_all_broadcast(dst, src, nbytes, …, UPC_PULL);
```

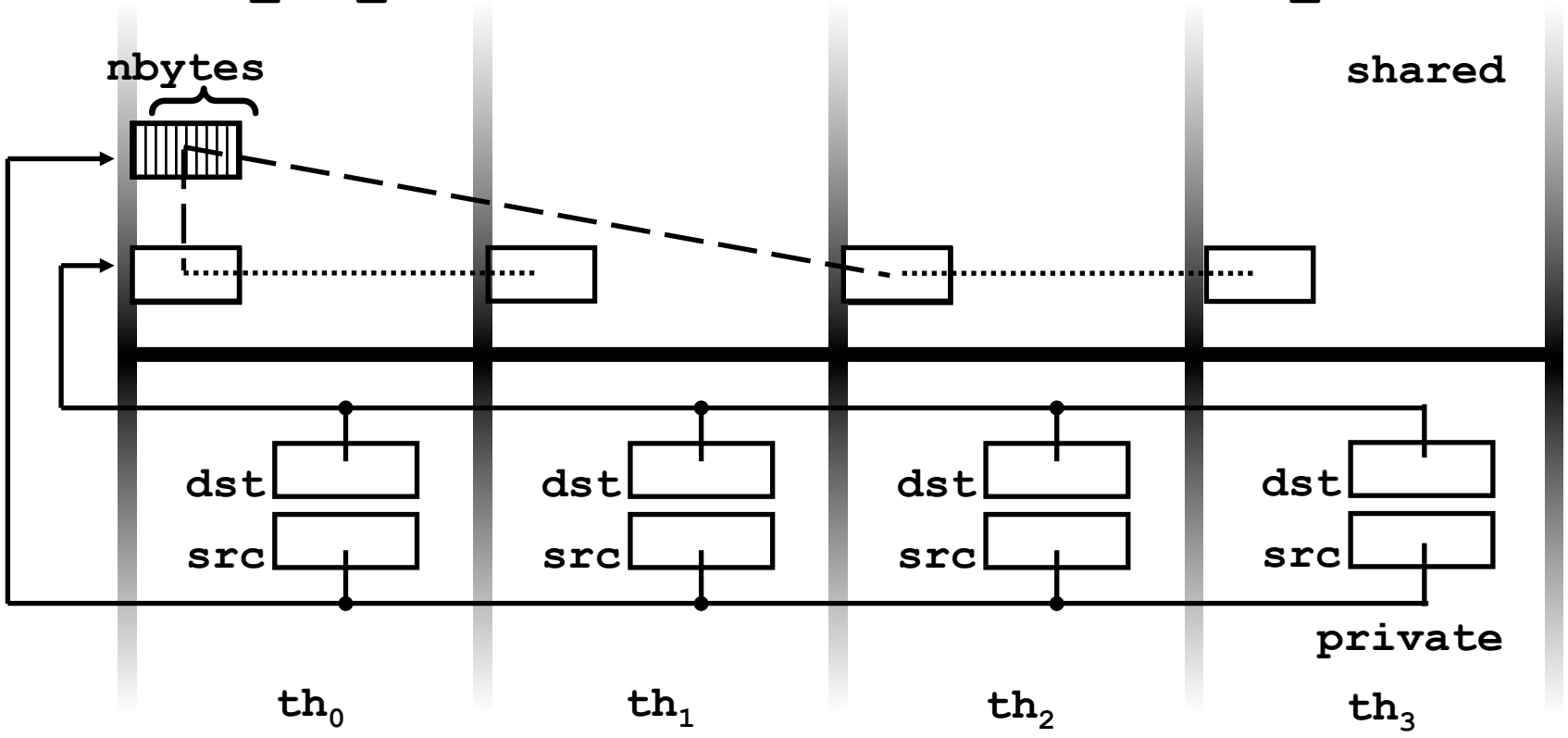


Algoritmo Multi-core Aware (1)

- Aumento del número de cores disponibles en los microprocesadores actuales
  - Desarrollo de algoritmos que sean conscientes de este hecho para lograr una mayor eficiencia
- El algoritmo Multi-core Aware se basa en designar a un thread como root en cada nodo, que se encargará de realizar la operación a nivel de nodo y de comunicarse con el root de la función

Algoritmo Multi-core Aware (2)

```
shared [] char src[nbytes];
shared [nbytes] char dst[nbytes * THREADS];
void xupc_all_broadcast(dst, src, nbytes, …, UPC_MULTICORE);
```



Algoritmo Multi-core Aware (y 3)

- Colectivas de Movimiento de Datos
  - **`xupc_all_broadcast`**
  - **`xupc_all_scatter`**
  - **`xupc_all_gather`**
  - **`xupc_all_gather_all`**
- Colectivas Computacionales
  - **`xupc_all_reduce_all`** (22 funciones)

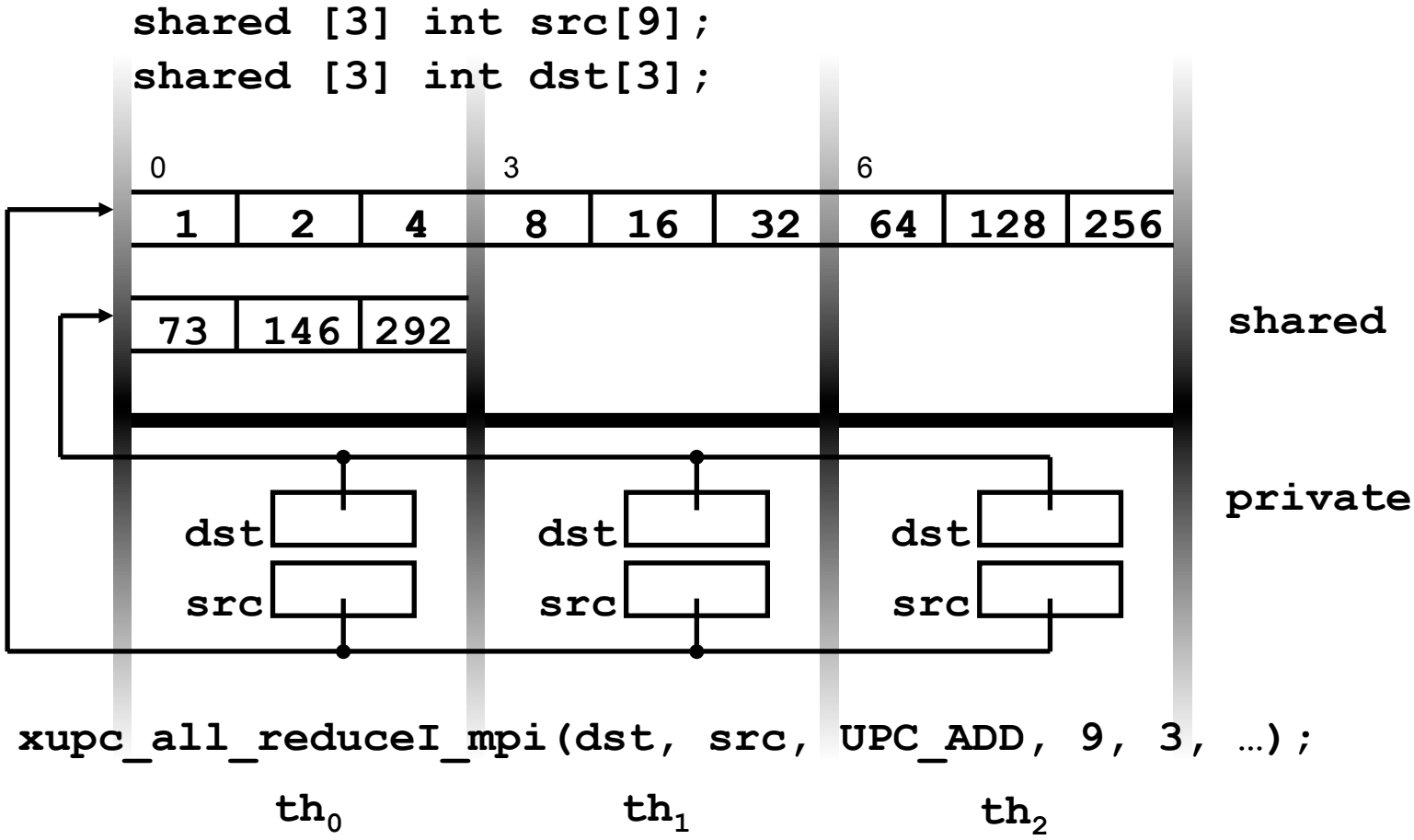
Motivación del Trabajo (3)

Limitaciones	Solución	Disponibilidad	Trabajo
No optimizadas para sistemas multi-core			
Funcionalidad reducida	Parcialmente	Parcialmente	
Arrays en memoria compartida	Value-Based	Parcialmente	
Mismo tamaño en todos los threads Datos al principio de cada bloque	Vector-Variant	Parcialmente	

Nuevas Funcionalidades en Colectivas en UPC (1)

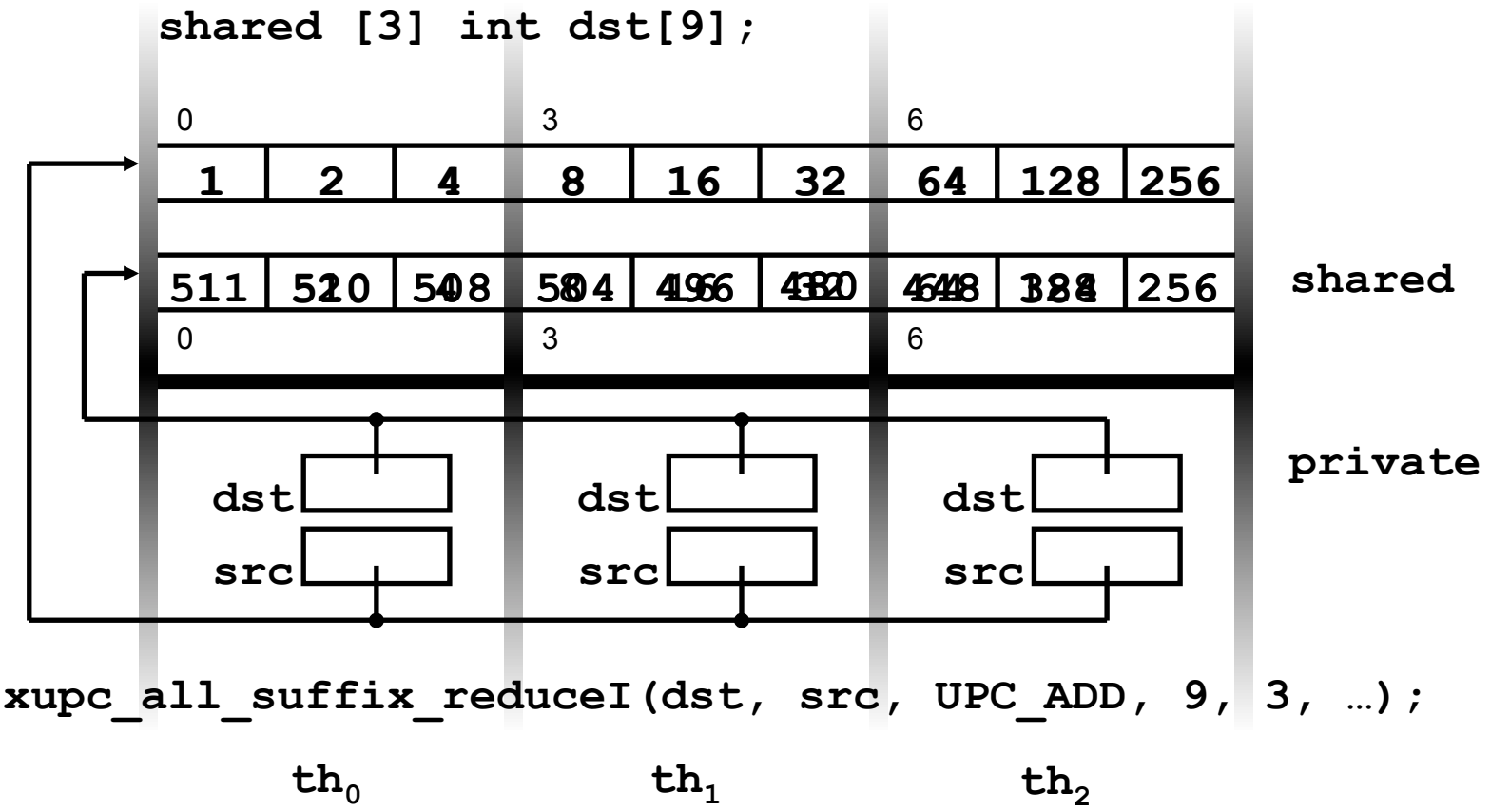
- Colectivas Computacionales
  - **`xupc_all_reduce_all`** (22 funciones)
  - **`xupc_all_reduce_mpi`** (22 funciones)
  - **`xupc_all_reduce_all_mpi`** (22 funciones)
  - **`xupc_all_suffix_reduce`** (22 funciones)
- Colectivas de Movimiento de Datos
  - **`xupc_all_broadcast_in_place`**

Nuevas Funcionalidades en Colectivas en UPC (2)



Nuevas Funcionalidades en Colectivas en UPC (3)

```
shared [3] int src[9];
shared [3] int dst[9];
```

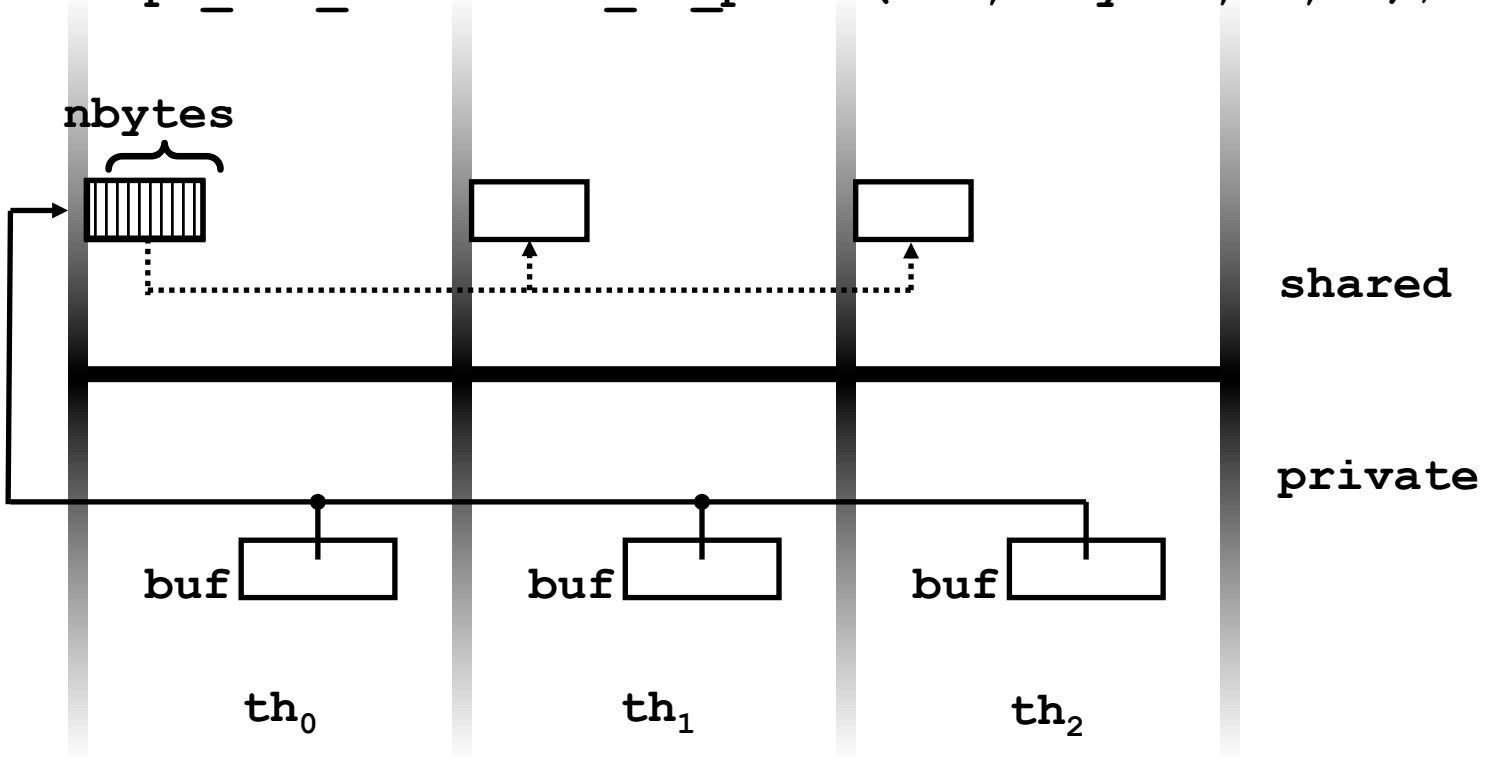



```
xupc_all_suffix_reduceI(dst, src, UPC_ADD, 9, 3, …);
```

th_0 th_1 th_2

Nuevas Funcionalidades en Colectivas en UPC (y 4)

```
shared [] char src[nbytes];
shared [nbytes] char dst[nbytes * THREADS];
void xupc_all_broadcast_in_place(buf, nbytes, 0, …);
```



Motivación del Trabajo (4)

Limitaciones	Solución	Disponibilidad	Trabajo
No optimizadas para sistemas multi-core			
Funcionalidad reducida	Parcialmente	Parcialmente	
Arrays en memoria compartida	Value-Based	Parcialmente	
Mismo tamaño en todos los threads Datos al principio de cada bloque	Vector-Variant	Parcialmente	

Array of Values Based Collectives (1)

- Funciones basadas en arrays de valores (privados)
- Trabajar en memoria compartida cuando no es necesario penaliza rendimiento
- Mejora programabilidad
  - El uso de operaciones colectivas requiere espacio shared
  - La biblioteca desarrollada libera al desarrollador de la escritura repetitiva de código semejante
    - Menor esfuerzo a realizar
    - Menor probabilidad de cometer errores
  - Envoltorio sobre colectivas estándar

Array of Values Based Collectives (2)

Código en UPC estándar	Código utilizando `xupc_colectiveav.h`
`declaración de array privado`	`declaración de array privado`
`uso del array`	`uso del array`
`reserva de espacio compartido`	
`copia de datos al espacio compartido`	
`upc_all_broadcast(espacio compartido)`	`xupc_allav_broadcast(array)`
`copia del resultado al array privado`	
`liberación del espacio compartido`	
`uso del array con el resultado`	`uso del array con el resultado`

Array of Values Based Collectives (y 3)

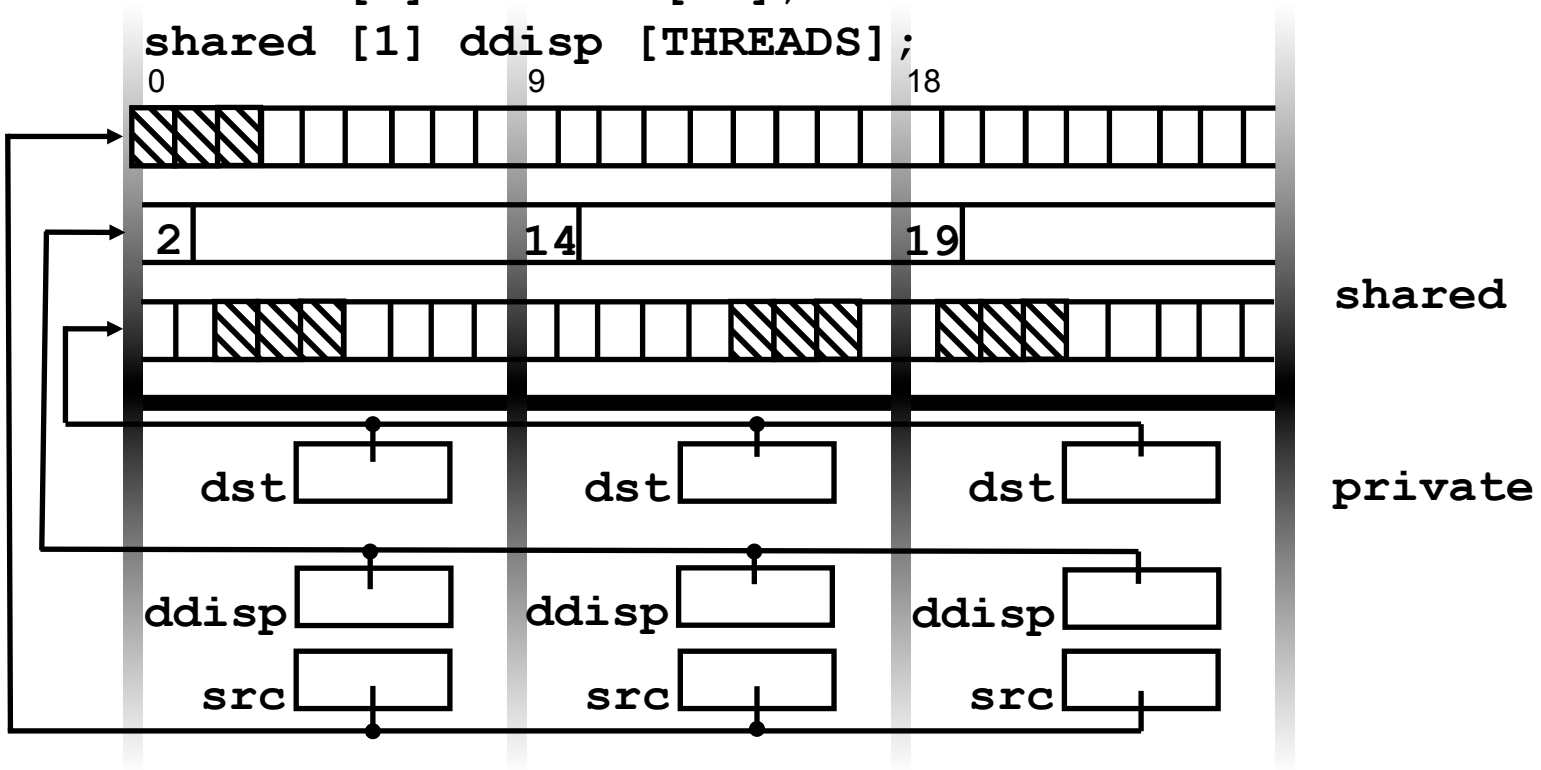
- Colectivas de Movimiento de Datos
  - **`xupc_allav_broadcast`**
  - **`xupc_allav_gather`**
  - **`xupc_allav_gather_all`**
  - **`xupc_allav_scatter`**
  - **`xupc_allav_permute`**
- Colectivas Computacionales
  - **`xupc_allav_reduce`**
  - **`xupc_allav_reduce_all`**
  - **`xupc_allav_prefix_reduce`**

Motivación del Trabajo (5)

Limitaciones	Solución	Disponibilidad	Trabajo
No optimizadas para sistemas multi-core			
Funcionalidad reducida	Parcialmente	Parcialmente	
Arrays en memoria compartida	Value-Based	Parcialmente	
Mismo tamaño en todos los threads Datos al principio de cada bloque	Vector-Variant	Parcialmente	

Vector Variant Collectives (1)

```
shared [9] int src[27];
shared [9] int dst[27];
shared [1] ddisp [THREADS];
```



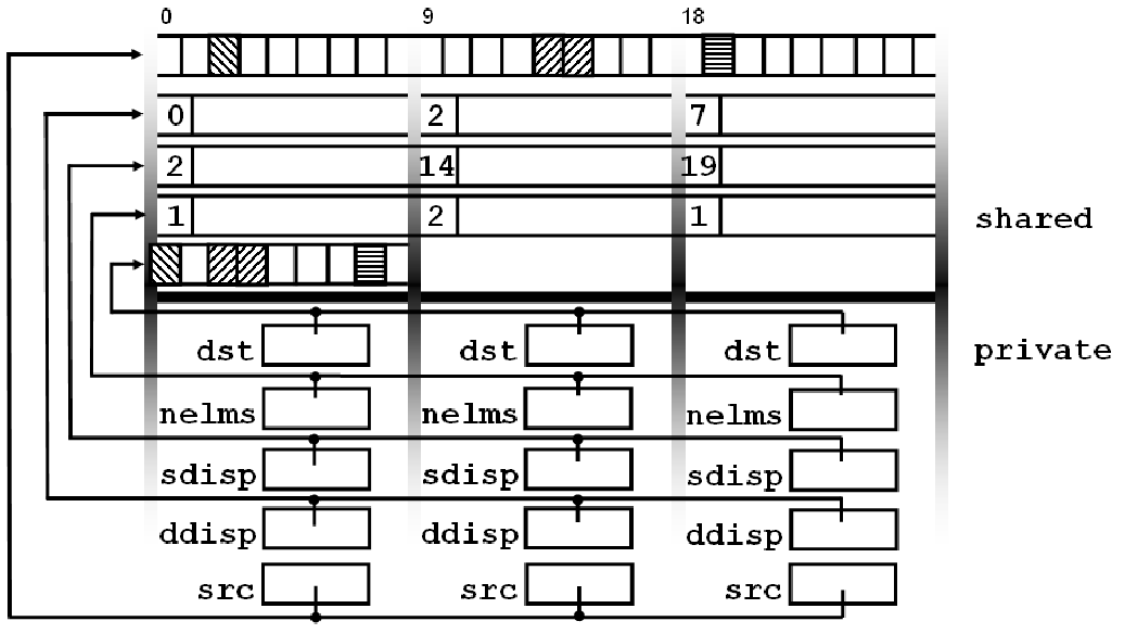

```
xupc_all_broadcast_v(dst, src, ddisp, 3, 9, sizeof(int), …);
```

th_0 th_1 th_2

Vector Variant Collectives (2)

```
shared [9] int src[27];
shared [9] int dst[9];
```

```
shared [1] ddisp [THREADS];
shared [1] sdisp [THREADS];
shared [1] nelems [THREADS];
```



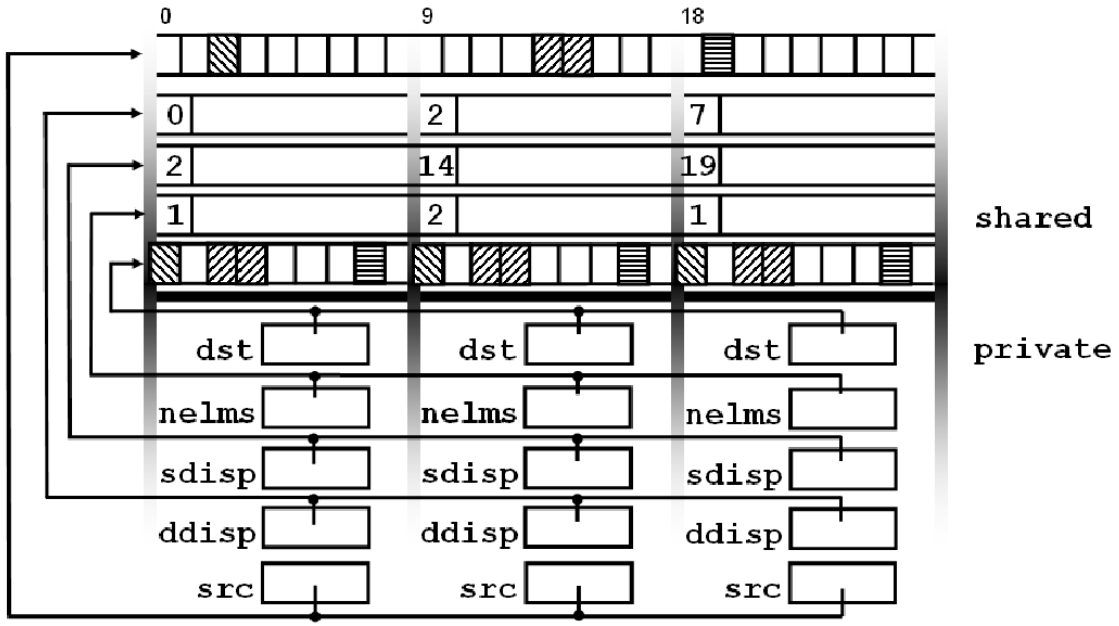

```
xupc_all_gather_v(dst,src,ddisp,sdisp,nelms,9,sizeof(int),…);
```

th_0 th_1 th_2

Vector Variant Collectives (y 3)

```
shared [9] int src[27];
shared [9] int dst[27];
```

```
shared [1] ddisp [THREADS];
shared [1] sdisp [THREADS];
shared [1] nelems [THREADS];
```




```
xupc_all_gather_all_v(dst,src,ddisp,sdisp,nelms,9,sizeof(int),…);
```

th_0 th_1 th_2

Motivación del Trabajo (y 6)

Limitaciones	Solución	Disponibilidad	Trabajo
No optimizadas para sistemas multi-core			
Funcionalidad reducida	Parcialmente	Parcialmente	
Arrays en memoria compartida	Value-Based	Parcialmente	
Mismo tamaño en todos los threads Datos al principio de cada bloque	Vector-Variant	Parcialmente	

Programabilidad y Productividad

- Otras aproximaciones a más bajo nivel explotan el conocimiento de la arquitectura: desarrollo muy costoso
- Aproximaciones a más alto nivel permiten un desarrollo más productivo
  - Cada vez es más costoso el desarrollo frente a la potencia computacional
  - Mayor programabilidad → Menor time-to-solution

NAS Parallel Benchmarks

		Sec. Fortran	MPI Fortran	Secuencial C	UPC	UPC colectiv
CG	nº líneas	506	1046	665	710	679
			106,72 %		6,77 %	2,11 %
	nº caracteres	16485	37501	16145	17200	16449
			127,48 %		6,53 %	1,88 %
EP	nº líneas	130	181	127	183	177
			39,23 %		44,09 %	39,37 %
	nº caracteres	4741	6567	2868	4117	3982
			38,52 %		43,55 %	38,84 %
FT	nº líneas	665	1278	575	1018	1002
			92,18 %		77,04 %	74,28 %
	nº caracteres	22188	44348	13090	21672	21331
			99,87 %		65,56 %	62,96 %
IS	nº líneas	353	627	353	528	505
			77,62 %		49,58 %	43,06 %
	nº caracteres	7273	13324	7273	13114	12572
			83,20 %		80,72 %	72,86 %
MG	nº líneas	885	1613	610	866	854
			82,26 %		41,97 %	40,00 %
	nº caracteres	27129	50497	14830	21990	21685
			86,14 %		48,28 %	46,22 %

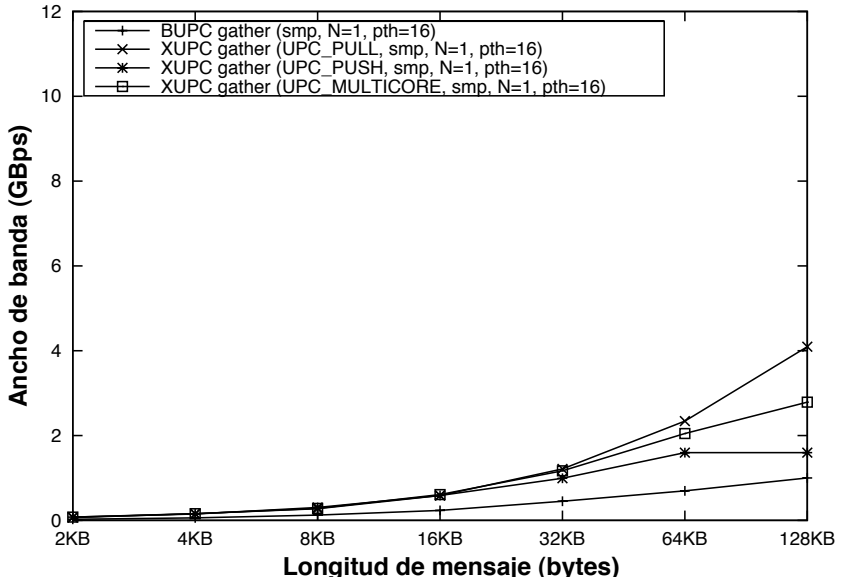
Rendimiento (1)

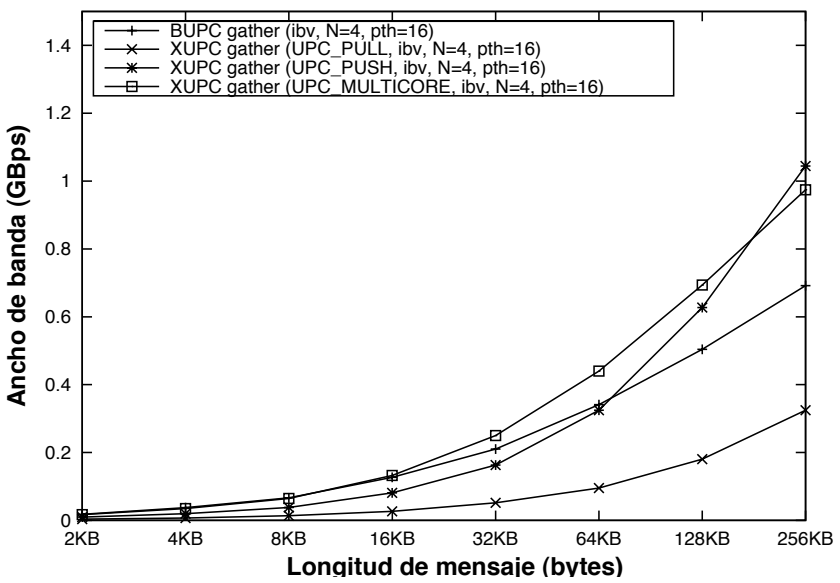
- Entorno de pruebas
  - Supercomputador FinisTerrae (CESGA)
  - 142 nodos HP Integrity rx7640
    - 8 procesadores Intel IA64 Itanium2 Montecito Dual Core a 1,6 GHz (16 cores) → 2272 cores
    - 128 GB de memoria →18 TB
  - 1 nodo HP IntegritySuperdome (128 cores y 1 TB)
  - 1 nodo HP IntegritySuperdome (128 cores y 384 GB)
  - Interconectados con red Infiniband 4x DDR a 20 Gbps
  - Sistema operativo SUSE Linux 10
    - Kernel 2.6.16.53-0.8_SFS2.3_0-default
  - Berkeley UPC v2.6.0, Intel C Compiler 10 y HP-MPI
  - 14 TFlops de rendimiento pico
  - #100 Top500 (noviembre 2007)

Rendimiento (2)

`xupc_all_gather`

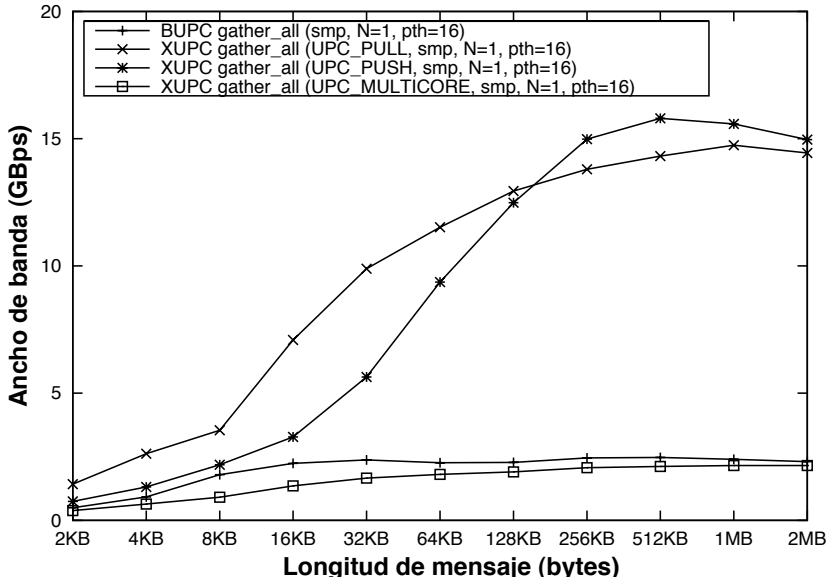
con 16 threads por nodo, utilizando 1 y 4 nodos

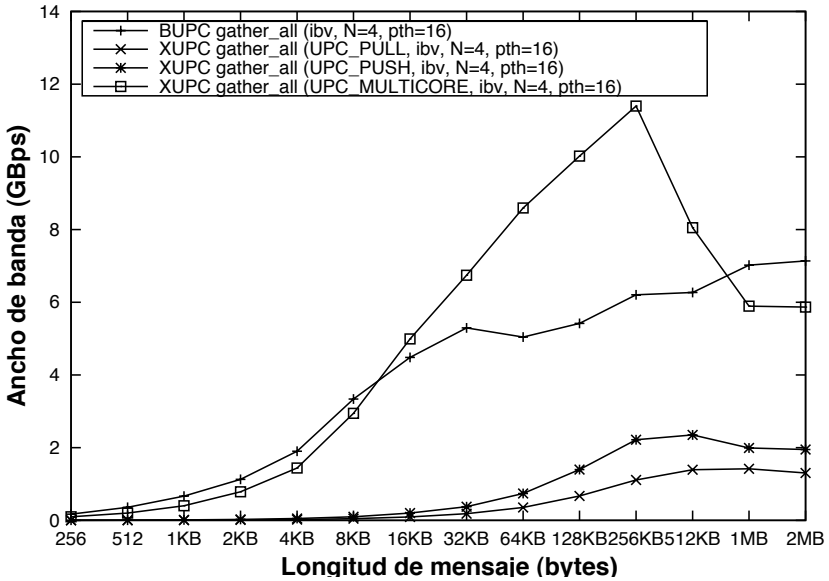




Rendimiento (3)

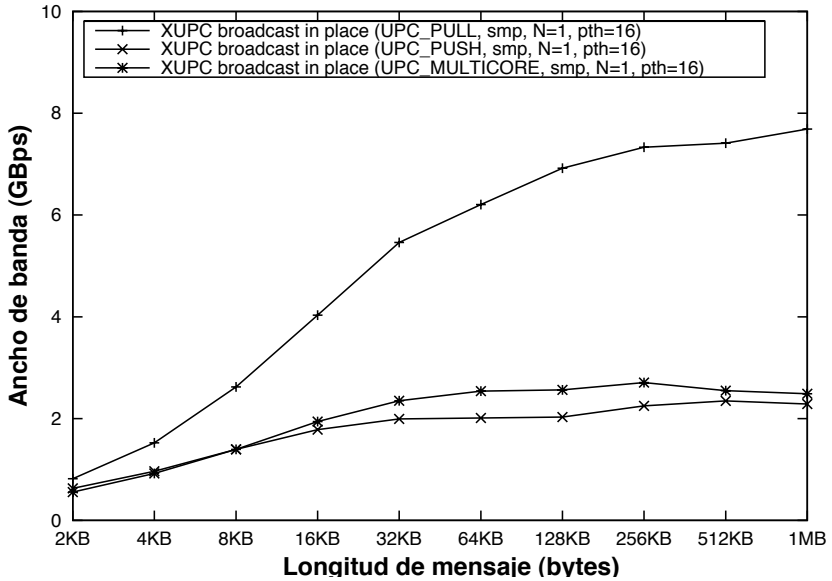
`xupc_all_gather_all`
con 16 threads por nodo, utilizando 1 y 4 nodos

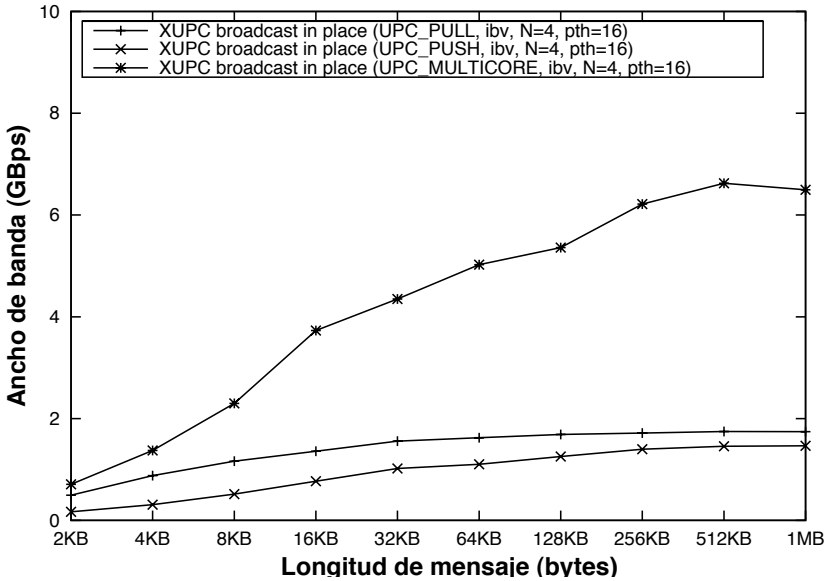




Rendimiento (y 4)

`xupc_all_broadcast_in_place`
con 16 threads por nodo, utilizando 1 y 4 nodos





Conclusiones y Principales Aportaciones (1)

- Revisión del estado actual de UPC para programación paralela y sus operaciones colectivas
  - Biblioteca estándar con importantes limitaciones en términos de eficiencia, funcionalidad y programabilidad
- Diseño e implementación del primer algoritmo multi-core aware para las colectivas en UPC
- Proporcionar 3 algoritmos por colectiva (Pull, Push y Multi-core Aware) fácilmente seleccionables
- Nuevas operaciones colectivas que favorecen la programabilidad y, por tanto, incrementan la productividad
  - Semejantes a las disponibles en MPI (aprovechamiento know-how)
  - Nuevas extensiones enfocadas a permitir una mayor flexibilidad

Conclusiones y Principales Aportaciones (y 2)

- Análisis en términos de programabilidad y productividad del lenguaje UPC y de las operaciones colectivas
- Evaluación del rendimiento de las funciones colectivas implementadas sobre el supercomputador Finis Terrae
- Una biblioteca extensa y optimizada de operaciones colectivas permite realizar movimientos de datos y operaciones de consolidación de forma eficiente y por medio de una única llamada a la función, lo que también favorece la programabilidad