Una Biblioteca Numérica Paralela para UPC

Jorge González-Domínguez[1]*, María J. Martín[1], Guillermo L. Taboada[1], Juan Touriño[1], Ramón Doallo[1], Andrés Gómez[2]

[1]Grupo de Arquitectura de Computadores
Universidad de A Coruña
{jgonzalezd,mariam,taboada,
juan,doallo}@udc.es

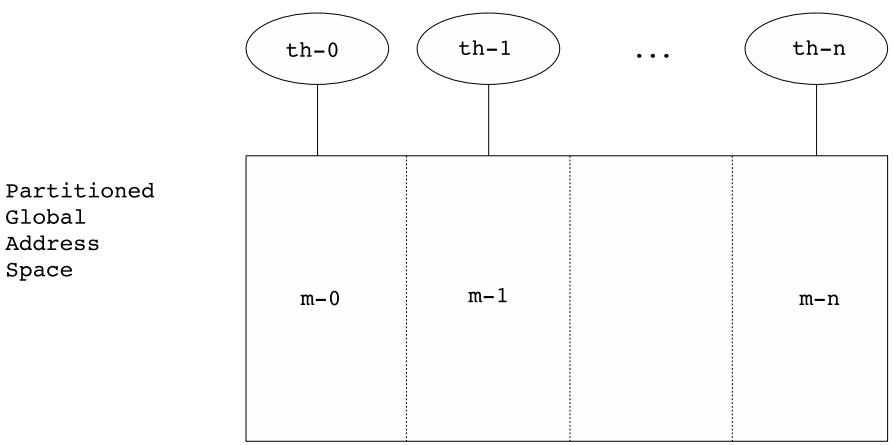
[2]Centro de Supercomputación de
Galicia (CESGA)
Santiago de Compostela
{agomez}@cesga.es

XX Jornadas de Paralelismo, Universidad de A Coruña

UPC: una Alternativa Adecuada para la Era Multi-core

Modelos de programación:

- Tradicionalmente: Memoria compartida o distribuida
- Reto: Arquitecturas con memoria híbrida
  - PGAS (Partitioned Global Address Space)



Lenguajes PGAS:

- UPC -> C
- Titanium -> Java
- Co-Array Fortran -> Fortran

Compiladores UPC:

- Berkeley UPC
- GCC (Intrepid)
- Michigan TU
- Compiladores de HP, IBM y Cray

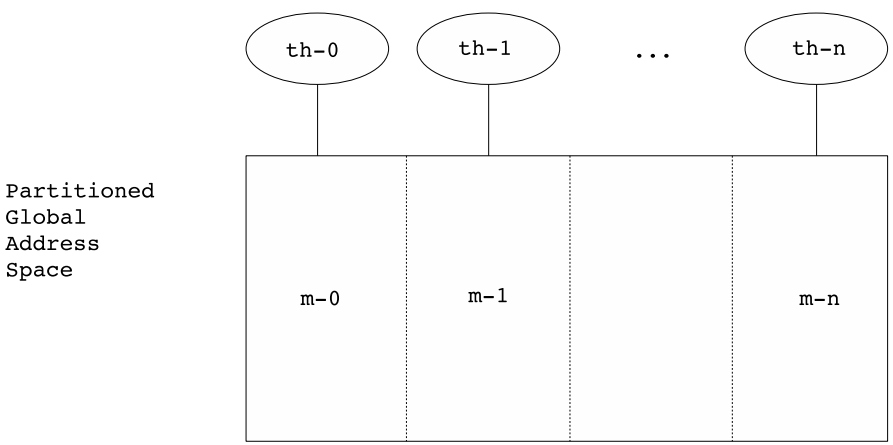
UPC: una Alternativa Adecuada para la Era Multi-core

Modelos de programación:

- Tradicionalmente: Memoria compartida o distribuida
- Reto: Arquitecturas con memoria híbrida
  - PGAS (Partitioned Global Address Space)

Lenguajes PGAS:

- UPC -> C
- Titanium -> Java
- Co-Array Fortran -> Fortran



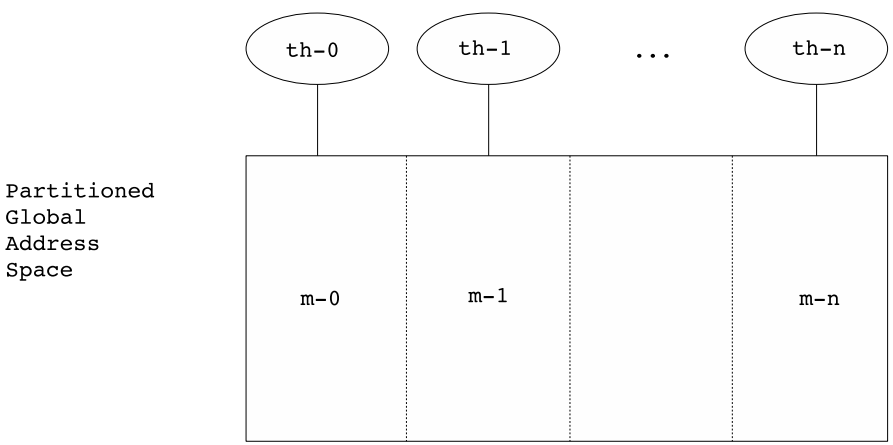
Compiladores UPC:

- Berkeley UPC
- GCC (Intrepid)
- Michigan TU
- Compiladores de HP, IBM y Cray

UPC: una Alternativa Adecuada para la Era Multi-core

Modelos de programación:

- Tradicionalmente: Memoria compartida o distribuida
- Reto: Arquitecturas con memoria híbrida
  - PGAS (Partitioned Global Address Space)



Lenguajes PGAS:

- UPC -> C
- Titanium -> Java
- Co-Array Fortran -> Fortran

Compiladores UPC:

- Berkeley UPC
- GCC (Intrepid)
- Michigan TU
- Compiladores de HP, IBM y Cray

Identificadores importantes

- `THREADS` -> Número total de threads en ejecución
- `MYTHREAD` -> Número del thread actual

```
#include<stdio.h>
#include<upc.h>
 int main() {
   printf("Thread %d de %d: Hola Mundo\n",
          MYTHREAD, THREADS);}
```

```
$ upcc -o holamundo holamundo.upc

$ upcrun -n 3 holamundo
 Thread 0 de 3: Hola Mundo
 Thread 2 de 3: Hola Mundo
 Thread 1 de 3: Hola Mundo
```

Identificadores importantes

- `THREADS` -> Número total de threads en ejecución
- `MYTHREAD` -> Número del thread actual

```
#include<stdio.h>
#include<upc.h>
 int main() {
   printf("Thread %d de %d: Hola Mundo\n",
          MYTHREAD, THREADS);}

 $ upcc -o holamundo holamundo.upc

 $ upcrun -n 3 holamundo
  Thread 0 de 3: Hola Mundo
  Thread 2 de 3: Hola Mundo
  Thread 1 de 3: Hola Mundo
```

Identificadores importantes

- `THREADS` -> Número total de threads en ejecución
- `MYTHREAD` -> Número del thread actual

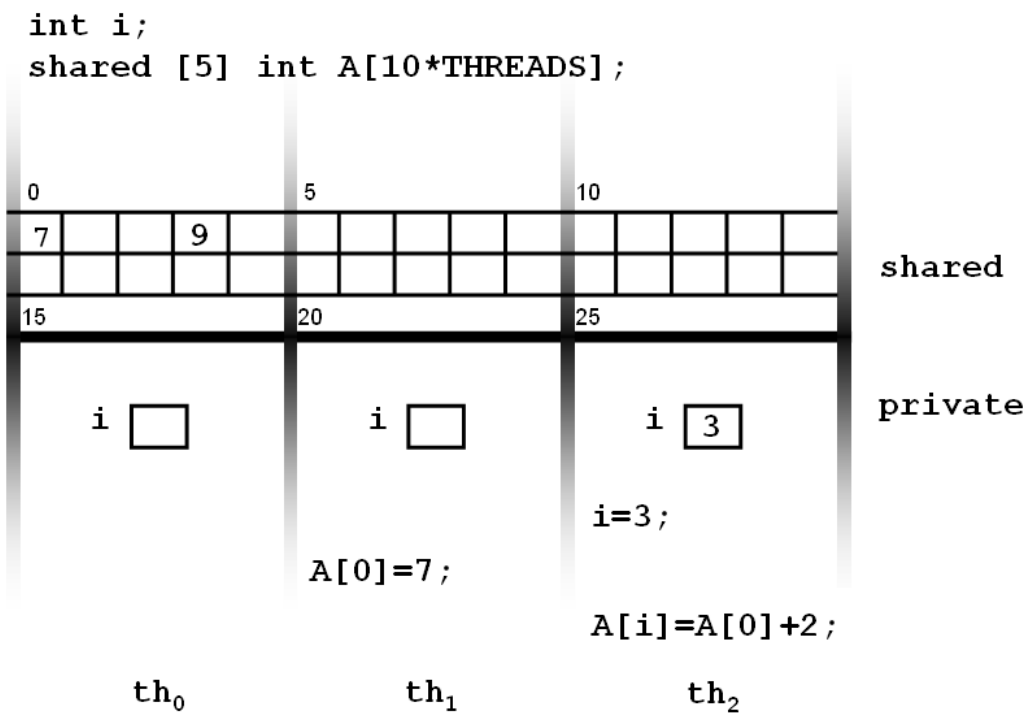
```
#include<stdio.h>
#include<upc.h>
 int main() {
   printf("Thread %d de %d: Hola Mundo\n",
          MYTHREAD, THREADS);}

 $ upcc -o holamundo holamundo.upc

 $ upcrun -n 3 holamundo
  Thread 0 de 3: Hola Mundo
  Thread 2 de 3: Hola Mundo
  Thread 1 de 3: Hola Mundo
```

Declaración de arrays compartidos

- `shared [block_factor] A [size]`
- `size` -> Tamaño del array
- `block_factor` -> Número de elementos consecutivos afines al mismo thread -> Tamaño de los trozos



Bibliotecas BLAS

- Basic Linear Algebra Subprograms
- Especificación de un conjunto de funciones numéricas
- Muy usada por científicos e ingenieros
- SparseBLAS y PBLAS (BLAS paralelo)

Implementaciones BLAS

- Genéricas y código abierto
  - GSL -> GNU
- Optimizadas para ciertas arquitecturas
  - MKL -> Intel
  - ACML -> AMD
  - CXML -> Compaq
  - MLIB -> HP

Bibliotecas BLAS

- Basic Linear Algebra Subprograms
- Especificación de un conjunto de funciones numéricas
- Muy usada por científicos e ingenieros
- SparseBLAS y PBLAS (BLAS paralelo)

Implementaciones BLAS

- Genéricas y código abierto
  - GSL -> GNU
- Optimizadas para ciertas arquitecturas
  - MKL -> Intel
  - ACML -> AMD
  - CXML -> Compaq
  - MLIB -> HP

Nivel BLAS	TnombreBLAS	Acción
BLAS1	Tcopy	Copia de un vector
	Tswap	Intercambio de los elementos de dos vectores
	Tscal	Multiplicación de un vector por un escalar
	Taxpy	Actualización de un vector usando los datos de otro: $y = \alpha * x + y$
	Tdot	Producto escalar de dos vectores
	Tnrm2	Norma euclídea
	Tasum	Suma del valor absoluto de todos los elementos de un vector
	iTamax	Encuentra el índice del vector con el valor máximo
	iTamin	Encuentra el índice del vector con el valor mínimo
BLAS2	Tgemv	Producto matriz-vector
	Ttrsv	Resolución de un sistema triangular de ecuaciones lineales
	Tger	Producto tensorial de dos vectores
BLAS3	Tgemm	Producto de dos matrices
	Ttrsm	Resolución de un bloque de sistemas triangulares de ecuaciones lineales

Computación numérica en UPC

- **Ninguna biblioteca numérica para lenguajes PGAS**

Alternativas del programador:

- Implementar las rutinas por si mismo
  - Mayor esfuerzo
  - Peor rendimiento
- Cambiar a un modelo de programación con bibliotecas numéricas
  - Memoria distribuida -> MPI
  - Memoria compartida -> OpenMP

Consecuencia:

- Barrera a la productividad de los lenguajes PGAS

Computación numérica en UPC

- **Ninguna biblioteca numérica para lenguajes PGAS**

Alternativas del programador:

- Implementar las rutinas por si mismo
  - Mayor esfuerzo
  - Peor rendimiento
- Cambiar a un modelo de programación con bibliotecas numéricas
  - Memoria distribuida -> MPI
  - Memoria compartida -> OpenMP

Consecuencia:

- Barrera a la productividad de los lenguajes PGAS

Computación numérica en UPC

- **Ninguna biblioteca numérica para lenguajes PGAS**

Alternativas del programador:

- Implementar las rutinas por si mismo
  - Mayor esfuerzo
  - Peor rendimiento
- Cambiar a un modelo de programación con bibliotecas numéricas
  - Memoria distribuida -> MPI
  - Memoria compartida -> OpenMP

Consecuencia:

- Barrera a la productividad de los lenguajes PGAS

Análisis de trabajos previos

Aproximación con memoria distribuida (Parallel -MPI- BLAS)

- Paradigma de Paso de Mensajes
- Sólo memoria privada
- Nuevas estructuras para representar matrices y vectores distribuidos
  - Difíciles de entender
- Funciones para ayudar a trabajar con ellas
  - Creación
  - Inicialización de los datos
  - Eliminación

Nueva aproximación

Uso de los arrays compartidos de UPC

Análisis de trabajos previos

Aproximación con memoria distribuida (Parallel -MPI- BLAS)

- Paradigma de Paso de Mensajes
- Sólo memoria privada
- Nuevas estructuras para representar matrices y vectores distribuidos
  - Difíciles de entender
- Funciones para ayudar a trabajar con ellas
  - Creación
  - Inicialización de los datos
  - Eliminación

Nueva aproximación

Uso de los arrays compartidos de UPC

Análisis de trabajos previos

Aproximación con memoria distribuida (Parallel -MPI- BLAS)

- Paradigma de Paso de Mensajes
- Sólo memoria privada
- Nuevas estructuras para representar matrices y vectores distribuidos
  - Difíciles de entender
- Funciones para ayudar a trabajar con ellas
  - Creación
  - Inicialización de los datos
  - Eliminación

Nueva aproximación

Uso de los arrays compartidos de UPC

Dos funciones por cada rutina BLAS

Funciones privadas

- Datos de entrada y salida en memoria privada
- Distribución de los datos interna a la función -> ni escogida ni conocida por el usuario

Funciones compartidas

- Datos de entrada y salida en memoria compartida
- Distribución de datos elegida por el usuario

Dos funciones por cada rutina BLAS

Funciones privadas

- Datos de entrada y salida en memoria privada
- Distribución de los datos interna a la función -> ni escogida ni conocida por el usuario

Funciones compartidas

- Datos de entrada y salida en memoria compartida
- Distribución de datos elegida por el usuario

Dos funciones por cada rutina BLAS

Funciones privadas

- Datos de entrada y salida en memoria privada
- Distribución de los datos interna a la función -> ni escogida ni conocida por el usuario

Funciones compartidas

- Datos de entrada y salida en memoria compartida
- Distribución de datos elegida por el usuario

`upc_blas_[p]Tnombreblas`

valor de `p`

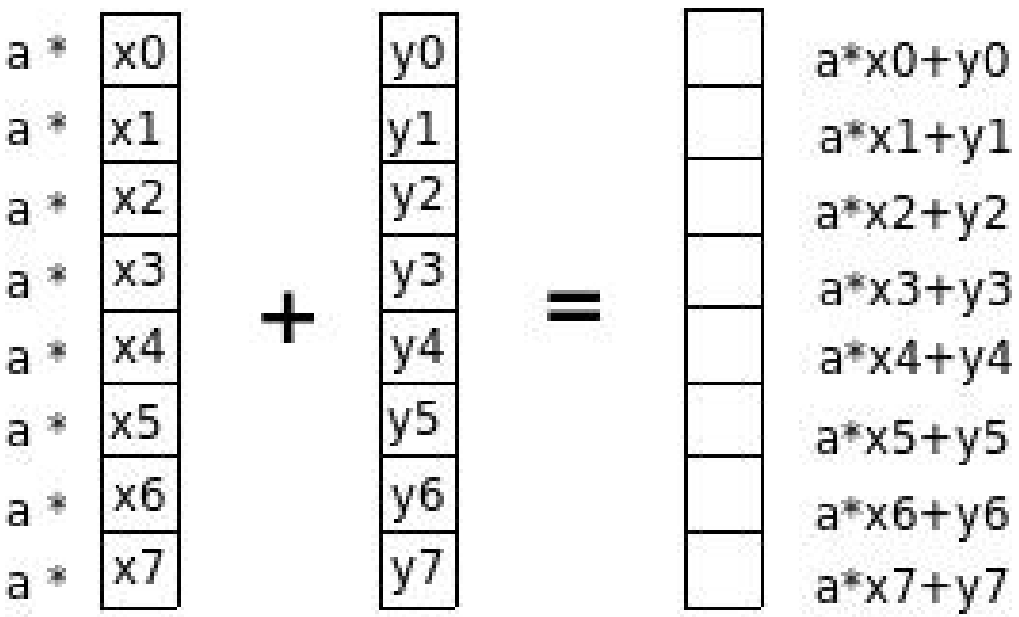
- _ -> versión compartida
- p -> versión privada

valor de `T`

- i -> integer
- l -> long
- f -> float
- d -> double

2 versiones *4 tipos de datos *14 rutinas = 112 funciones

$$y = a * x + y$$



versión privada -> `upc_blas_pdaxpy`
versión compartida ->`upc_blas_daxpy`

```
int upc_blas_pdaxpy(const int size, const
double a, const int thread_src, const double
*x, const int thread_dst, double *y);
```

Parameters

- `size`. Tamaño del vector
- `a`. Factor a multiplicar
- `x, y`. Punteros privados a las posiciones de la memoria privada donde se almacenan los vectores
- `thread_src`. [0,`THREADS`]
  - Thread con los vectores de entrada `x` e `y` en su memoria privada
    - Si THREADS -> Vectores replicados en todas las memorias privadas
- `thread_dst`. [0,`THREADS`]
  - Thread con la memoria privada donde se escribirá el vector de salida
    - Si THREADS -> Salida replicada en todas las memorias privadas -> BROADCAST

```
int upc_blas_pdaxpy(const int size, const
double a, const int thread_src, const double
*x, const int thread_dst, double *y);
```

Parameters

- `size`. Tamaño del vector
- `a`. Factor a multiplicar
- `x, y`. Punteros privados a las posiciones de la memoria privada donde se almacenan los vectores
- `thread_src`. [0,`THREADS`]
  - Thread con los vectores de entrada `x` e `y` en su memoria privada
  - Si THREADS -> Vectores replicados en todas las memorias privadas
- `thread_dst`. [0,`THREADS`]
  - Thread con la memoria privada donde se escribirá el vector de salida
  - Si THREADS -> Salida replicada en todas las memorias privadas -> BROADCAST

```
int upc_blas_pdaxpy(const int size, const
double a, const int thread_src, const double
*x, const int thread_dst, double *y);
```

Parameters

- `size`. Tamaño del vector
- `a`. Factor a multiplicar
- `x, y`. Punteros privados a las posiciones de la memoria privada donde se almacenan los vectores
- `thread_src`. [0,`THREADS`]
  - Thread con los vectores de entrada `x` e `y` en su memoria privada
  - Si THREADS -> Vectores replicados en todas las memorias privadas
- `thread_dst`. [0,`THREADS`]
  - Thread con la memoria privada donde se escribirá el vector de salida
  - Si THREADS -> Salida replicada en todas las memorias privadas -> BROADCAST

```
int upc_blas_pdaxpy(const int size, const
double a, const int thread_src, const double
*x, const int thread_dst, double *y);
```

Parameters

- `size`. Tamaño del vector
- `a`. Factor a multiplicar
- `x, y`. Punteros privados a las posiciones de la memoria privada donde se almacenan los vectores
- `thread_src`. [0,`THREADS`]
  - Thread con los vectores de entrada `x` e `y` en su memoria privada
  - Si THREADS -> Vectores replicados en todas las memorias privadas
- `thread_dst`. [0,`THREADS`]
  - Thread con la memoria privada donde se escribirá el vector de salida
  - Si THREADS -> Salida replicada en todas las memorias privadas -> BROADCAST

```
int upc_blas_pdaxpy(const int size, const
double a, const int thread_src, const double
*x, const int thread_dst, double *y);
```

Parameters

- `size`. Tamaño del vector
- `a`. Factor a multiplicar
- `x, y`. Punteros privados a las posiciones de la memoria privada donde se almacenan los vectores
- `thread_src`. [0,`THREADS`]
  - Thread con los vectores de entrada `x` e `y` en su memoria privada
  - Si THREADS -> Vectores replicados en todas las memorias privadas
- `thread_dst`. [0,`THREADS`]
  - Thread con la memoria privada donde se escribirá el vector de salida
  - Si THREADS -> Salida replicada en todas las memorias privadas -> BROADCAST

```
int upc_blas_pdaxpy(const int size, const
double a, const int thread_src, const double
*x, const int thread_dst, double *y);
```

Parameters

- `size`. Tamaño del vector
- `a`. Factor a multiplicar
- `x, y`. Punteros privados a las posiciones de la memoria privada donde se almacenan los vectores
- `thread_src`. [0,`THREADS`]
  - Thread con los vectores de entrada `x` e `y` en su memoria privada
  - Si THREADS -> Vectores replicados en todas las memorias privadas
- `thread_dst`. [0,`THREADS`]
  - Thread con la memoria privada donde se escribirá el vector de salida
  - Si THREADS -> Salida replicada en todas las memorias privadas -> BROADCAST

```
int upc_blas_daxpy(const int block_size, const
int size, const double a, shared const double
*x, shared double *y);
```

Parameters

- `size`. Tamaño de los vectores -> Igual que en privado
- `a`. Factor a multiplicar -> Igual que en privado
- `x, y`. Punteros privados a las posiciones de la memoria compartida donde se almacenan los vectores

```
int upc_blas_daxpy(const int block_size, const
int size, const double a, shared const double
*x, shared double *y);
```

Parameters

- `size`. Tamaño de los vectores -> Igual que en privado
- `a`. Factor a multiplicar -> Igual que en privado
- `x, y`. Punteros privados a las posiciones de la memoria compartida donde se almacenan los vectores

```
int upc_blas_daxpy(const int block_size, const
int size, const double a, shared const double
*x, shared double *y);
```

Parameters

- `size`. Tamaño de los vectores -> Igual que en privado
- `a`. Factor a multiplicar -> Igual que en privado
- `x, y`. Punteros privados a las posiciones de la memoria compartida donde se almacenan los vectores

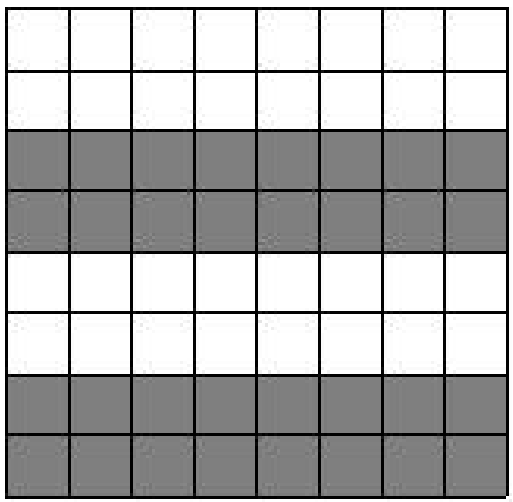
Significado de `block_size` para vectores

- In the range [1,`size`]
- Cantidad de elementos consecutivos afines al mismo thread
- En la práctica, `block_size` = block_factor del array compartido
- Determina la distribución del trabajo

`shared [block_size] y [size]`

Significado de `block_size` para las matrices distribuidas por filas

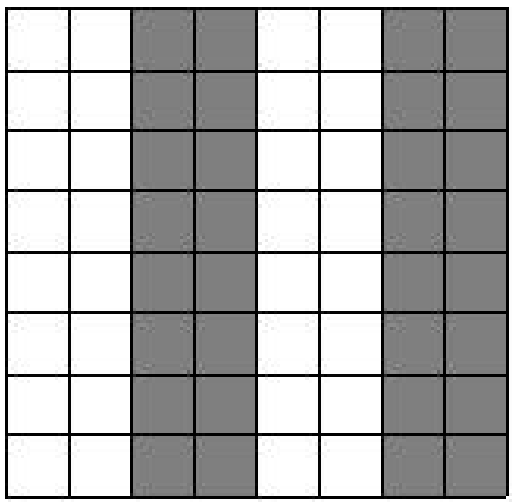
- Parámetro adicional `dist_dimm = row_dist`
- En el intervalo [1,`rows`]
- Cantidad de filas consecutivas afines al mismo thread
- Determina la distribución del trabajo



`shared [block_size*cols] y [rows*cols]`

Significado de `block_size` para matrices distribuidas por columnas

- Parámetro adicional `dist_dimm = col_dist`
- En el intervalo [1,`cols`]
- Cantidad de columnas consecutivas afines al mismo thread
- Determina la distribución del trabajo



`shared [block_size] y [rows*cols]`

Técnicas para obtener buena eficiencia

- Técnicas de optimización de UPC:
  - Privatización de los accesos a memoria compartida

Técnicas para obtener buena eficiencia

- Técnicas de optimización de UPC:
  - Privatización de los accesos a memoria compartida

Punteros privados a memoria privada

- Punteros de C estándar
- Almacenados en memoria privada
- Pueden acceder:
  - Memoria privada
  - Parte de la memoria compartida afín al thread
- Accesos muy rápidos

Punteros privados a memoria compartida

- Almacenados en memoria privada
- Pueden acceder:
  - Toda la memoria compartida
- Más pesados que los punteros de C -> Accesos más lentos

Técnicas para obtener buena eficiencia

- Técnicas de optimización de UPC:
  - Privatización de los accesos a memoria compartida
  - Agrupación de accesos a memoria compartida remota (`upc_memget, upc_memput, upc_memcpy`)
  - Solapamiento de accesos remotos con computación
- Distribución correcta de la carga de trabajo y los datos entre los threads -> versión privada
- Llamadas internas a bibliotecas numéricas paralelas muy eficientes

Técnicas para obtener buena eficiencia

- Técnicas de optimización de UPC:
  - Privatización de los accesos a memoria compartida
  - Agrupación de accesos a memoria compartida remota (`upc_memget, upc_memput, upc_memcpy`)
  - Solapamiento de accesos remotos con computación
- Distribución correcta de la carga de trabajo y los datos entre los threads -> versión privada
- Llamadas internas a bibliotecas numéricas paralelas muy eficientes

Técnicas para obtener buena eficiencia

- Técnicas de optimización de UPC:
  - Privatización de los accesos a memoria compartida
  - Agrupación de accesos a memoria compartida remota (`upc_memget, upc_memput, upc_memcpy`)
  - Solapamiento de accesos remotos con computación
- Distribución correcta de la carga de trabajo y los datos entre los threads -> versión privada
- Llamadas internas a bibliotecas numéricas paralelas muy eficientes

Técnicas para obtener buena eficiencia

- Técnicas de optimización de UPC:
  - Privatización de los accesos a memoria compartida
  - Agrupación de accesos a memoria compartida remota (`upc_memget, upc_memput, upc_memcpy`)
  - Solapamiento de accesos remotos con computación
- Distribución correcta de la carga de trabajo y los datos entre los threads -> versión privada
- Llamadas internas a bibliotecas numéricas paralelas muy eficientes

Finis Terrae (CESGA)

142 nodos HP Integrity rx7640, cada uno:

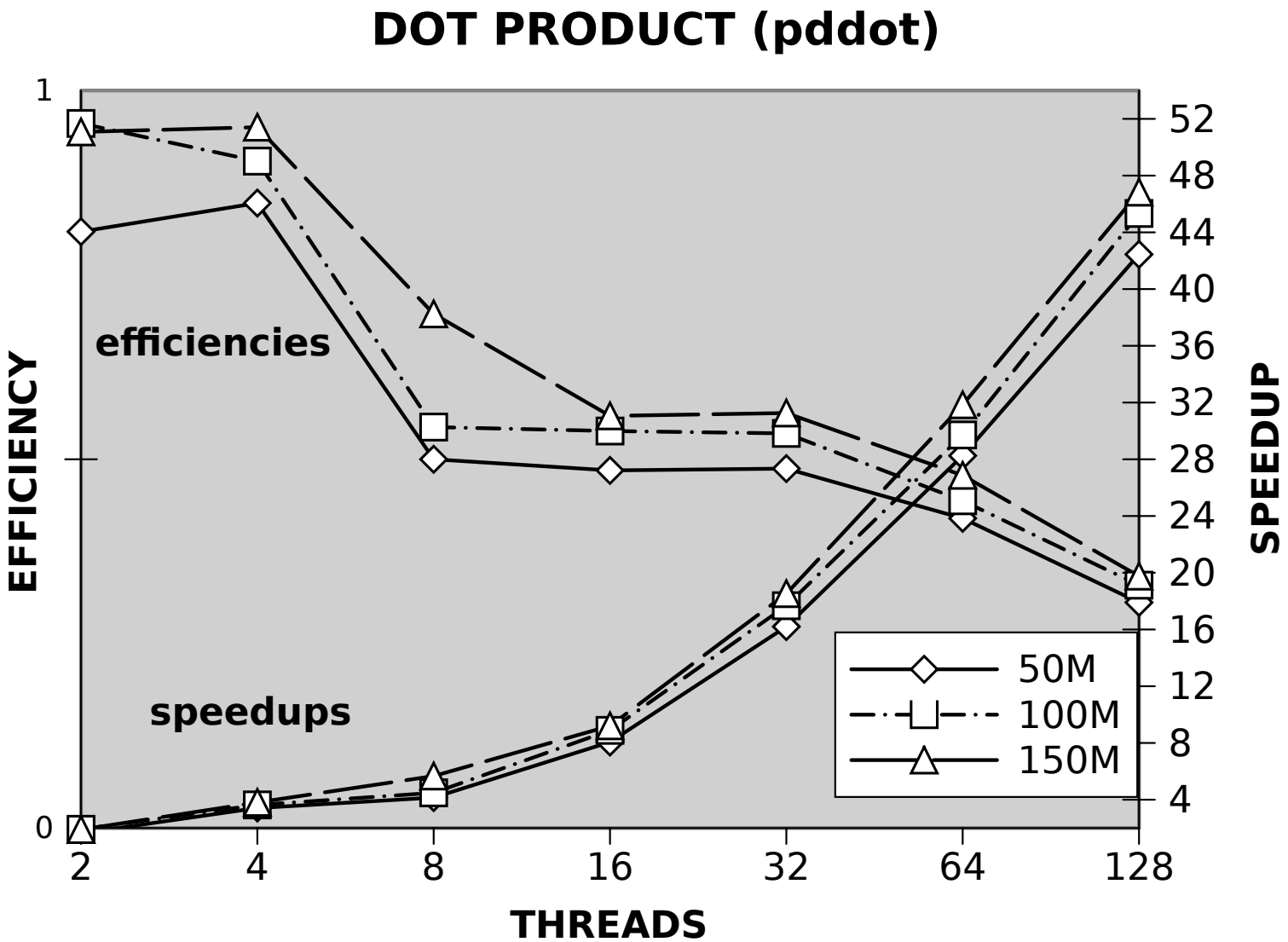
- 16 núcleos Montvale Itanium2 (IA64) at 1.6 GHz
  - 2 celdas, cada una
    - 4 procesadores dual-core
    - 1 módulo de memoria compartida
- 128 GB RAM
- Mellanox InfiniBand HCA (16 Gbps bandwidth)

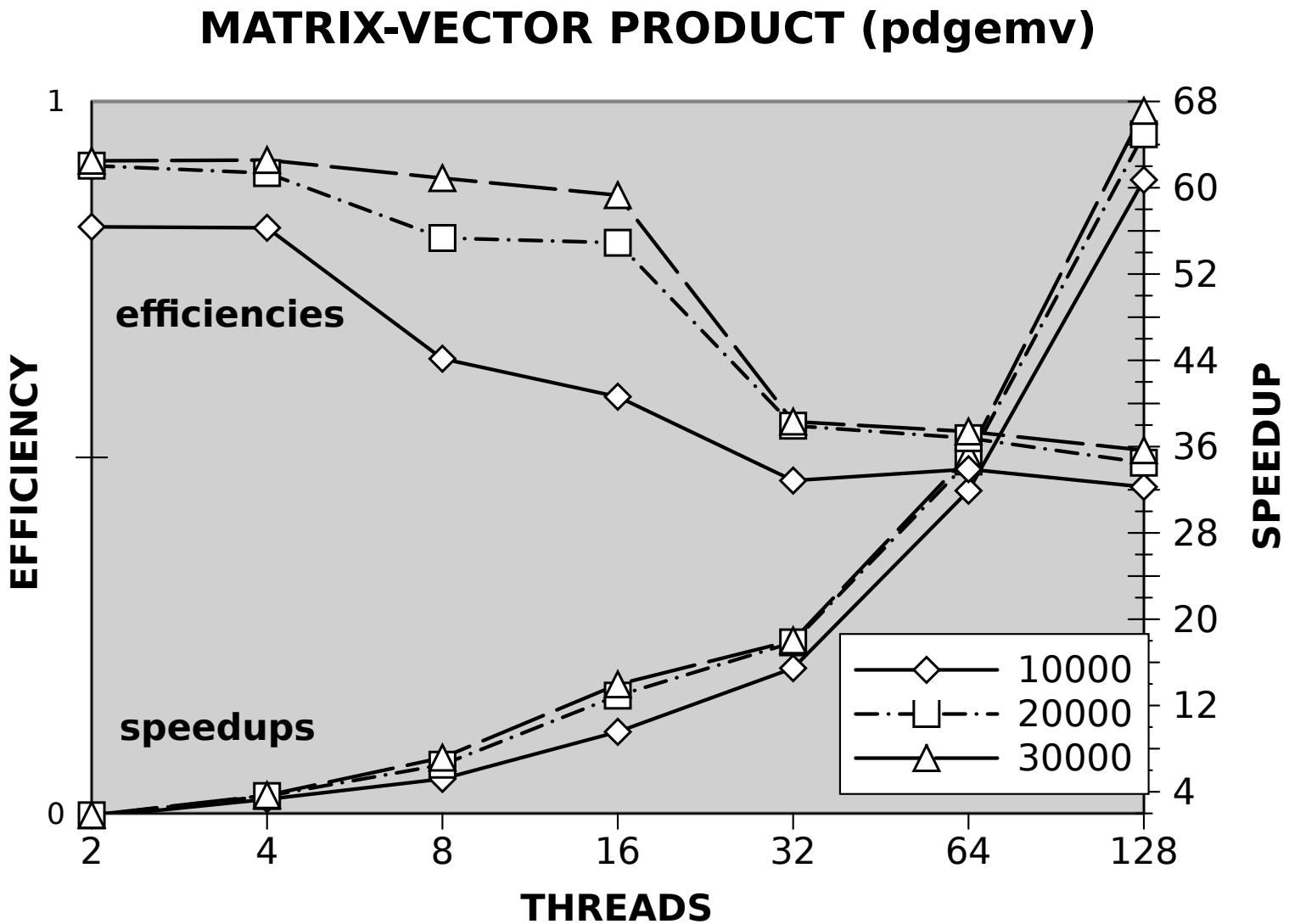
SW Configuration:

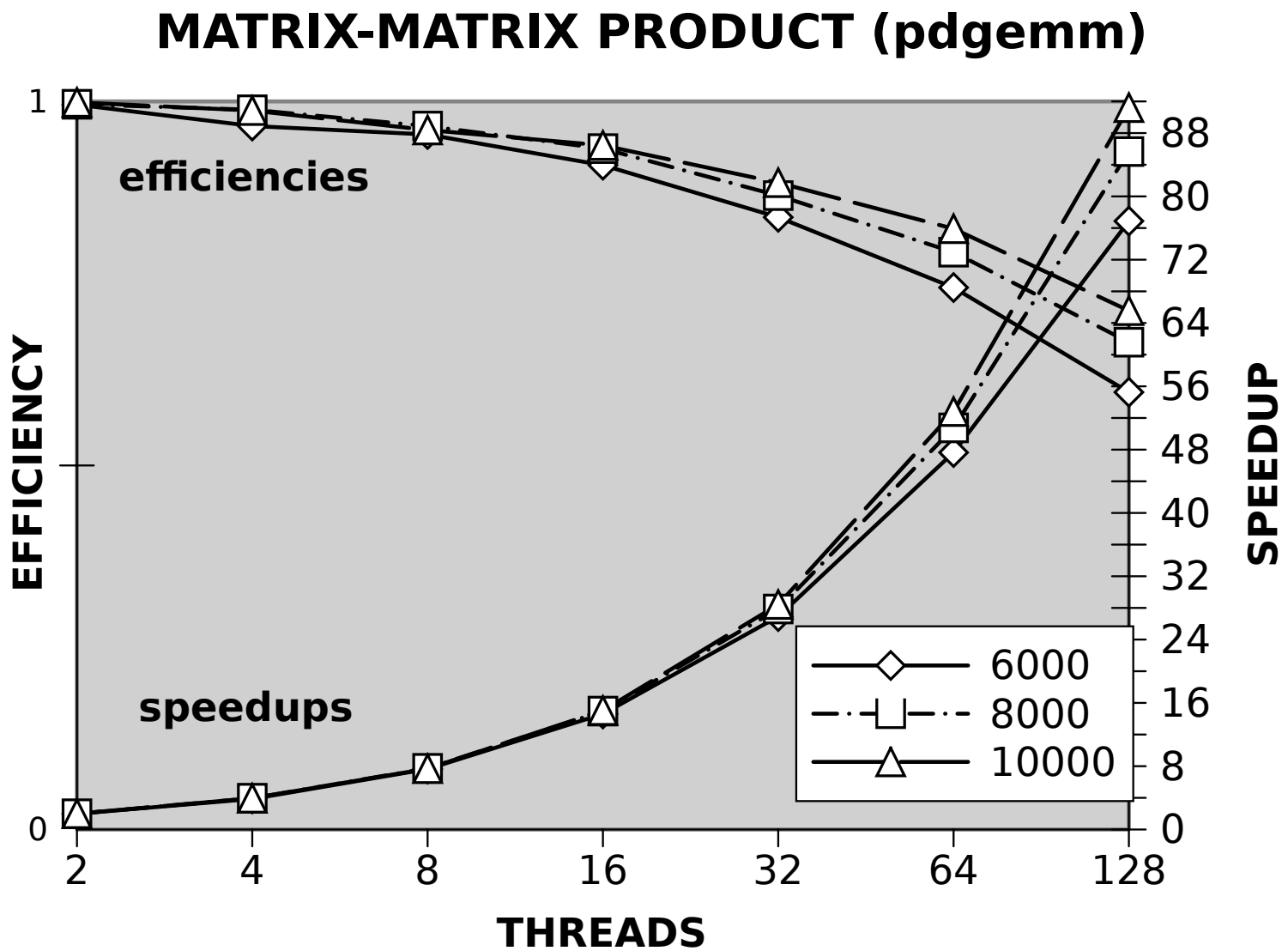
- Berkeley UPC (BUPC) 2.6
- Intel Math Kernel Library (MKL) 9.1
  - Todas las funciones BLAS1, BLAS2 y BLAS3 secuenciales
  - Otras rutinas: SparseBLAS, LAPACK, ScaLAPACK...

Configuración de los benchmarks

- Configuración de memoria híbrida
  - Explotación de la localidad de los threads en el mismo nodo -> memoria compartida
  - Aumenta la escalabilidad -> memoria distribuida
- 4 threads por nodo, 2 por celda
- Versión privada
- `src_threads = THREADS`
- `dst_threads = 0`


DOT PRODUCT (pddot)
EFFICIENCY
SPEEDUP
THREADS
efficiencies
speedups
50M
100M
150M
1
0
2
4
8
16
32
64
128
52
48
44
40
36
32
28
24
20
16
12
8
4


MATRIX-VECTOR PRODUCT (pdgemv)
EFFICIENCY
SPEEDUP
THREADS
efficiencies
speedups
1
0
68
60
52
44
36
28
20
12
4
2
4
8
16
32
64
128
10000
20000
30000


MATRIX-MATRIX PRODUCT (pdgemm)
efficiencies
speedups
EFFICIENCY
SPEEDUP
THREADS
1
0
2
4
8
16
32
64
128
88
80
72
64
56
48
40
32
24
16
8
0
6000
8000
10000

Summary

- Primera biblioteca numérica desarrollada para UPC -> Novedad
- Permite almacenar los datos de entrada y/o salida en memoria privada o compartida -> Flexibilidad
- Usa funciones BLAS secuenciales -> Portabilidad
- Scalabilidad demostrada con test experimentales -> Eficiencia

Trabajo futuro

Desarrollo de una biblioteca de computación numérica dispersa para UPC

Summary

- Primera biblioteca numérica desarrollada para UPC -> Novedad
- Permite almacenar los datos de entrada y/o salida en memoria privada o compartida -> Flexibilidad
- Usa funciones BLAS secuenciales -> Portabilidad
- Scalabilidad demostrada con test experimentales -> Eficiencia

Trabajo futuro

Desarrollo de una biblioteca de computación numérica dispersa para UPC

¿Preguntas?

Contacto: Jorge González-Domínguez *jgonzalezd@udc.es*

Grupo de Arquitectura de Computadores, Departamento de Electrónica y Sistemas

Universidad de A Coruña