A Parallel Numerical Library for UPC

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

[1]Computer Architecture Group
University of A Coruña (Spain)
{jgonzalezd,mariam,taboada,
juan,doallo}@udc.es

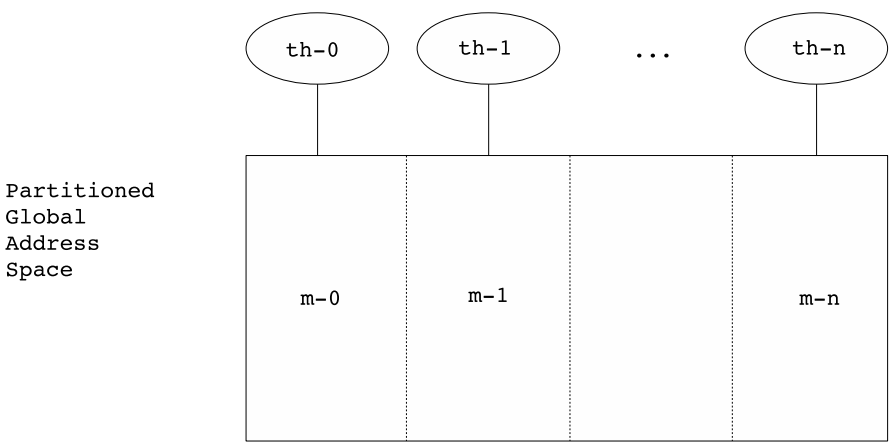
[2]Galicia Supercomputing Center
(CESGA)
Santiago de Compostela (Spain)
{agomez}@cesga.es

15th International European Conference on Parallel and Distributed Computing (Euro-Par 2009), Delft University of Technology, Delft, The Netherlands

UPC: a Suitable Alternative for HPC in Multi-core Era

Programming models:

- Traditionally: Shared/Distributed memory programming models
- Challenge: hybrid memory architectures
  - PGAS (Partitioned Global Address Space)



PGAS Languages:

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

UPC Compilers:

- Berkeley UPC
- GCC (Intrepid)
- Michigan TU
- HP, Cray and IBM UPC Compilers

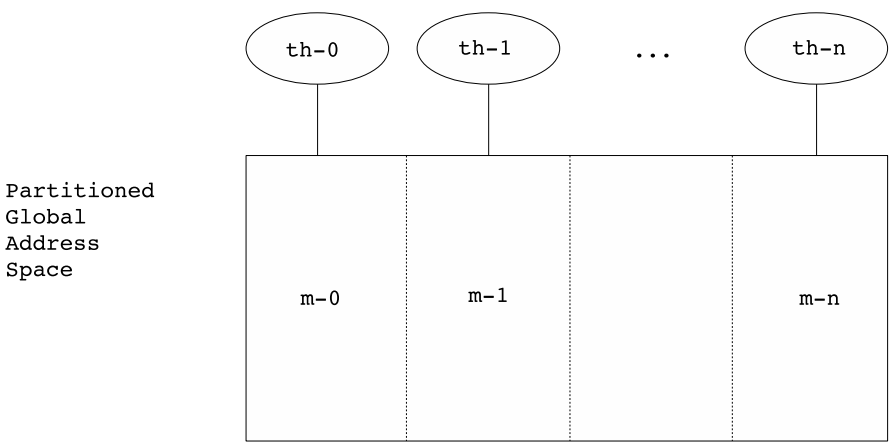
UPC: a Suitable Alternative for HPC in Multi-core Era

Programming models:

- Traditionally: Shared/Distributed memory programming models
- Challenge: hybrid memory architectures
  - PGAS (Partitioned Global Address Space)

PGAS Languages:

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



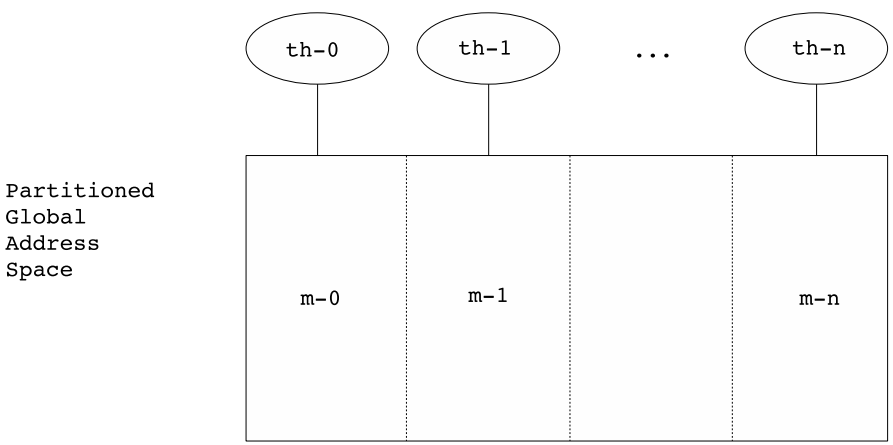
UPC Compilers:

- Berkeley UPC
- GCC (Intrepid)
- Michigan TU
- HP, Cray and IBM UPC Compilers

UPC: a Suitable Alternative for HPC in Multi-core Era

Programming models:

- Traditionally: Shared/Distributed memory programming models
- Challenge: hybrid memory architectures
  - PGAS (Partitioned Global Address Space)



PGAS Languages:

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

UPC Compilers:

- Berkeley UPC
- GCC (Intrepid)
- Michigan TU
- HP, Cray and IBM UPC Compilers

Important identifiers

- `THREADS` -> Total number of threads in execution
- `MYTHREAD` -> Rank of the current thread

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

```
$ upcc -o helloworld helloworld.upc

$ upcrun -n 3 helloworld
 Thread 0 of 3: Hello world
 Thread 2 of 3: Hello world
 Thread 1 of 3: Hello world
```

Important identifiers

- `THREADS` -> Total number of threads in execution
- `MYTHREAD` -> Rank of the current thread

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

 $ upcc -o helloworld helloworld.upc

 $ upcrun -n 3 helloworld
  Thread 0 of 3: Hello world
  Thread 2 of 3: Hello world
  Thread 1 of 3: Hello world
```

Important identifiers

- `THREADS` -> Total number of threads in execution
- `MYTHREAD` -> Rank of the current thread

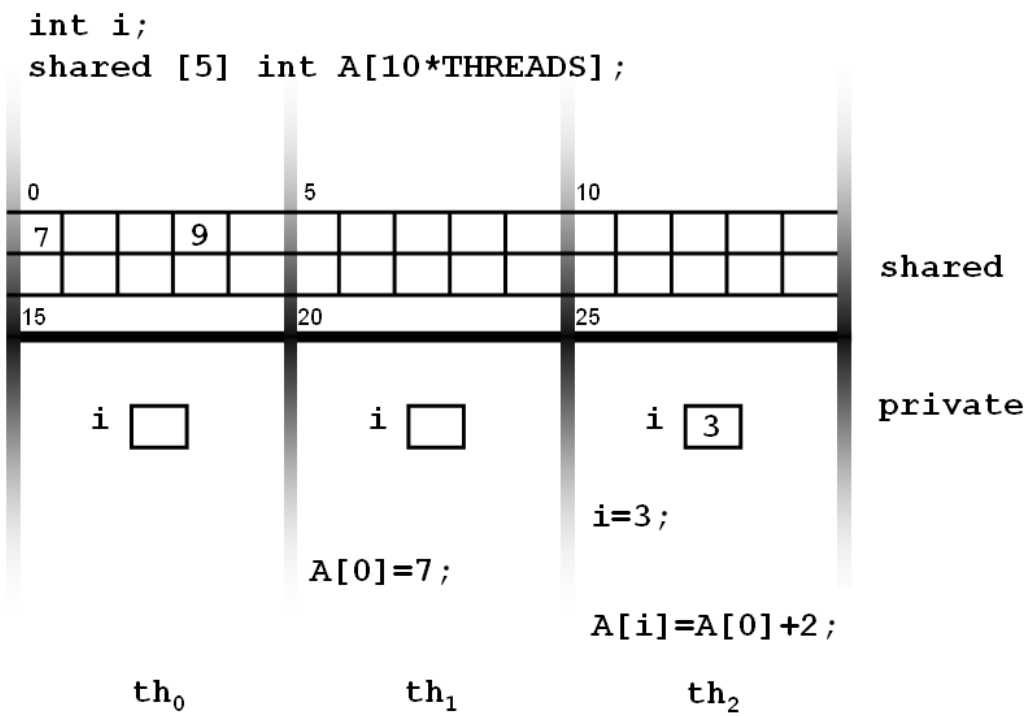
```
#include<stdio.h>
#include<upc.h>
 int main() {
   printf("Thread %d of %d: Hello world\n",
          MYTHREAD, THREADS);}

 $ upcc -o helloworld helloworld.upc

 $ upcrun -n 3 helloworld
  Thread 0 of 3: Hello world
  Thread 2 of 3: Hello world
  Thread 1 of 3: Hello world
```

Shared array declaration

- `shared [block_factor] A [size]`
- `size` -> Total number of elements
- `block_factor` -> Number of consecutive elements with affinity to the same thread -> Size of the chunks



BLAS libraries

- Basic Linear Algebra Subprograms
- Specification of a set of numerical functions
- Widely used by scientists and engineers
- SparseBLAS and PBLAS (Parallel BLAS)

BLAS implementations

- Generic and open source
  - GSL -> GNU
- Optimized for specific architectures
  - MKL -> Intel
  - ACML -> AMD
  - CXML -> Compaq
  - MLIB -> HP

BLAS libraries

- Basic Linear Algebra Subprograms
- Specification of a set of numerical functions
- Widely used by scientists and engineers
- SparseBLAS and PBLAS (Parallel BLAS)

BLAS implementations

- Generic and open source
  - GSL -> GNU
- Optimized for specific architectures
  - MKL -> Intel
  - ACML -> AMD
  - CXML -> Compaq
  - MLIB -> HP

BLAS level	Tblasname	Action
BLAS1	Tcopy	Copies a vector
	Tswap	Swaps the elements of two vectors
	Tscal	Scales a vector by a scalar
	Taxpy	Updates a vector using another one: $y = \alpha * x + y$
	Tdot	Dot product
	Tnrm2	Euclidean norm
	Tasum	Sums the absolute value of the elements of a vector
	iTamax	Finds the index with the maximum value
	iTamin	Finds the index with the minimum value
BLAS2	Tgemv	Matrix-vector product
	Ttrsv	Solves a triangular system of equations
	Tger	Outer product
BLAS3	Tgemm	Matrix-matrix product
	Ttrsm	Solves a block of triangular systems of equations

Numerical computing in UPC

- **No numerical libraries for PGAS languages**

Alternatives for the programmers:

- Develop the routine by themselves
  - More effort
  - Worse performance
- Use different programming models with parallel numerical libraries
  - Distributed memory -> MPI
  - Shared memory -> OpenMP

Consequence:

- Barrier to the productivity of PGAS languages.

Numerical computing in UPC

- **No numerical libraries for PGAS languages**

Alternatives for the programmers:

- Develop the routine by themselves
  - More effort
  - Worse performance
- Use different programming models with parallel numerical libraries
  - Distributed memory -> MPI
  - Shared memory -> OpenMP

Consequence:

- Barrier to the productivity of PGAS languages.

Numerical computing in UPC

- **No numerical libraries for PGAS languages**

Alternatives for the programmers:

- Develop the routine by themselves
  - More effort
  - Worse performance
- Use different programming models with parallel numerical libraries
  - Distributed memory -> MPI
  - Shared memory -> OpenMP

Consequence:

- Barrier to the productivity of PGAS languages.

Analysis of related works

Distributed memory approach (Parallel -MPI- BLAS)

- Message Passing paradigm
- Only private memory
- New structures to represent distributed vectors or matrices
  - Difficult to understand and work with
- Functions to help to work with them
  - Creation
  - Storage of data
  - Deletion

New approach

Usage of UPC shared arrays

Analysis of related works

Distributed memory approach (Parallel -MPI- BLAS)

- Message Passing paradigm
- Only private memory
- New structures to represent distributed vectors or matrices
  - Difficult to understand and work with
- Functions to help to work with them
  - Creation
  - Storage of data
  - Deletion

New approach

Usage of UPC shared arrays

Analysis of related works

Distributed memory approach (Parallel -MPI- BLAS)

- Message Passing paradigm
- Only private memory
- New structures to represent distributed vectors or matrices
  - Difficult to understand and work with
- Functions to help to work with them
  - Creation
  - Storage of data
  - Deletion

New approach

Usage of UPC shared arrays

Two functions for each BLAS routine

Private functions

- Input and output data in private memory
- Pointers from private to private
- Data distribution internal to the function -> not chosen or known by the user

Shared functions

- Input and output data in shared memory
- Pointers from private to shared
- Data distribution chosen by the user through a parameter

Two functions for each BLAS routine

Private functions

- Input and output data in private memory
- Pointers from private to private
- Data distribution internal to the function -> not chosen or known by the user

Shared functions

- Input and output data in shared memory
- Pointers from private to shared
- Data distribution chosen by the user through a parameter

Two functions for each BLAS routine

Private functions

- Input and output data in private memory
- Pointers from private to private
- Data distribution internal to the function -> not chosen or known by the user

Shared functions

- Input and output data in shared memory
- Pointers from private to shared
- Data distribution chosen by the user through a parameter

upc_blas_[p]Tblasname

p value

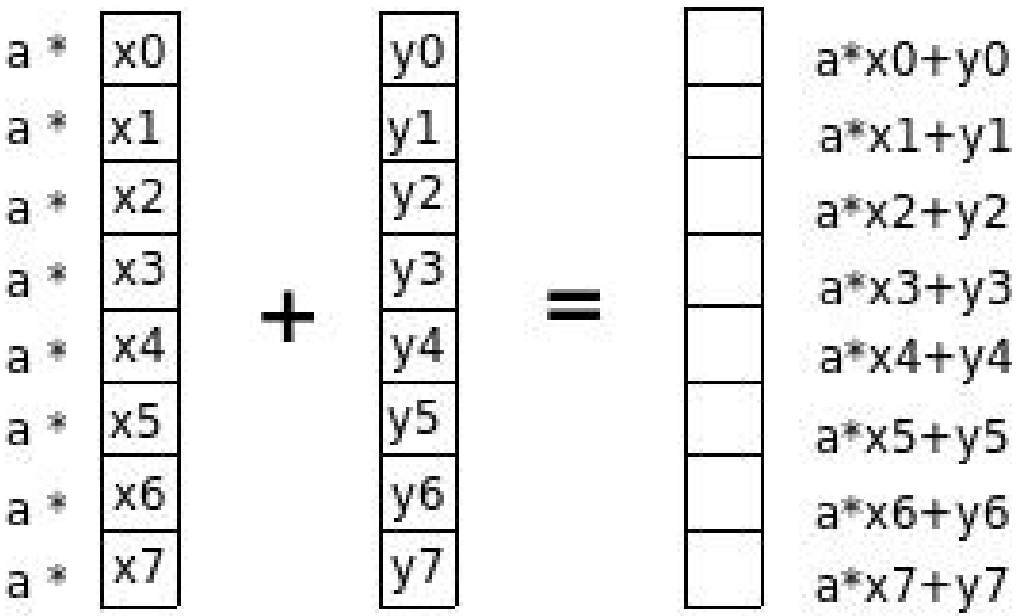
- _ -> shared version
- p -> private version

T value

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

2 versions *4 datatypes *14 routines = 112 functions

$$y = a * x + y$$



private version -> `upc_blas_pdaxpy`
shared version ->`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`. Vectors length
- `a`. Scale factor
- `x, y`. Private pointer to the position of private memory where vectors elements are stored
- `thread_src`. [0,`THREADS`]
  - Number of the thread with input `x` and `y` data in its private memory
  - If THREADS -> Vectors replicated in all private spaces
- `thread_dst`. [0,`THREADS`]
  - Number of the thread whith the private memory where output data will be written
  - If THREADS -> Output replicated in all private spaces -> 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`. Vectors length
- `a`. Scale factor
- `x, y`. Private pointer to the position of private memory where vectors elements are stored
- `thread_src`. [0,`THREADS`]
  - Number of the thread with input `x` and `y` data in its private memory
  - If THREADS -> Vectors replicated in all private spaces
- `thread_dst`. [0,`THREADS`]
  - Number of the thread whith the private memory where output data will be written
  - If THREADS -> Output replicated in all private spaces -> 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`. Vectors length
- `a`. Scale factor
- `x, y`. Private pointer to the position of private memory where vectors elements are stored
- `thread_src`. [0,`THREADS`]
  - Number of the thread with input `x` and `y` data in its private memory
  - If THREADS -> Vectors replicated in all private spaces
- `thread_dst`. [0,`THREADS`]
  - Number of the thread whith the private memory where output data will be written
  - If THREADS -> Output replicated in all private spaces -> 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`. Vectors length
- `a`. Scale factor
- `x, y`. Private pointer to the position of private memory where vectors elements are stored
- `thread_src`. [0,`THREADS`]
  - Number of the thread with input `x` and `y` data in its private memory
  - If THREADS -> Vectors replicated in all private spaces
- `thread_dst`. [0,`THREADS`]
  - Number of the thread whith the private memory where output data will be written
  - If THREADS -> Output replicated in all private spaces -> 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`. Vectors length
- `a`. Scale factor
- `x, y`. Private pointer to the position of private memory where vectors elements are stored
- `thread_src`. [0,`THREADS`]
  - Number of the thread with input `x` and `y` data in its private memory
  - If THREADS -> Vectors replicated in all private spaces
- `thread_dst`. [0,`THREADS`]
  - Number of the thread whith the private memory where output data will be written
  - If THREADS -> Output replicated in all private spaces -> 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`. Vectors length
- `a`. Scale factor
- `x, y`. Private pointer to the position of private memory where vectors elements are stored
- `thread_src`. [0,`THREADS`]
  - Number of the thread with input `x` and `y` data in its private memory
  - If THREADS -> Vectors replicated in all private spaces
- `thread_dst`. [0,`THREADS`]
  - Number of the thread whith the private memory where output data will be written
  - If THREADS -> Output replicated in all private spaces -> BROADCAST

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

Parameters

- `size`. Vectors length -> The same as private
- `a`. Scale factor -> The same as private
- `x, y`. Private pointer to the position of shared memory where vectors elements are stored

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

Parameters

- `size`. Vectors length -> The same as private
- `a`. Scale factor -> The same as private
- `x, y`. Private pointer to the position of shared memory where vectors elements are stored

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

Parameters

- `size`. Vectors length -> The same as private
- `a`. Scale factor -> The same as private
- `x, y`. Private pointer to the position of shared memory where vectors elements are stored

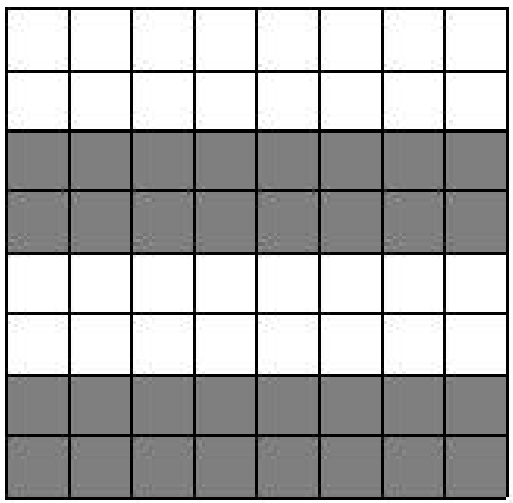
Meaning of `block_size` for vectors

- In the range [1,`size`]
- Number of consecutive elements with affinity to the same thread
- For performance, `block_size` = block_factor of shared arrays
- Determines the distribution of the work

```
shared [block_size] y [size]
```

Meaning of `block_size` for matrices distributed by rows

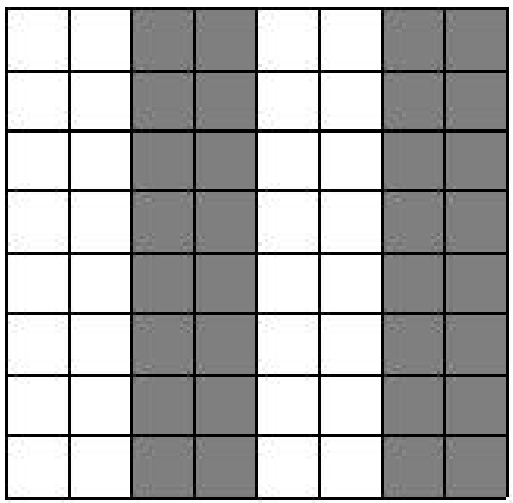
- Additional parameter `dist_dimm = row_dist`
- In the range [1,`rows`]
- Number of consecutive rows with affinity to the same thread
- Determines the distribution of the work



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

Meaning of `block_size` for matrices distributed by columns

- Additional parameter `dist_dimm = col_dist`
- In the range [1,`cols`]
- Number of consecutive columns with affinity to the same thread
- Determines the distribution of the work



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

General steps to achieve a good efficiency

- UPC optimization techniques:
  - Privatization of the accesses to shared memory

General steps to achieve a good efficiency

- UPC optimization techniques:
  - Privatization of the accesses to shared memory

Private pointers to private memory

- The standard C pointers
- Stored in private memory
- Able to access:
  - Private memory
  - Part of the shared memory with affinity to the thread
- Very fast to dereference

Private pointers to shared memory

- Stored in private memory
- Able to access:
  - Any position in all shared memory
- Heavier than standard C pointers -> Slower accesses

General steps to achieve a good efficiency

- UPC optimization techniques:
  - Privatization of the accesses to shared memory
  - Agregation of remote shared memory accesses (`upc_memget, upc_memput, upc_memcpy`)
  - Overlapping remote accesses with computation
- Correct distribution of the workload and data among threads -> private case
- Call the most efficient underlying numerical libraries

General steps to achieve a good efficiency

- UPC optimization techniques:
  - Privatization of the accesses to shared memory
  - Agregation of remote shared memory accesses (`upc_memget, upc_memput, upc_memcpy`)
  - Overlapping remote accesses with computation
- Correct distribution of the workload and data among threads -> private case
- Call the most efficient underlying numerical libraries

General steps to achieve a good efficiency

- UPC optimization techniques:
  - Privatization of the accesses to shared memory
  - Agregation of remote shared memory accesses (`upc_memget, upc_memput, upc_memcpy`)
  - Overlapping remote accesses with computation
- Correct distribution of the workload and data among threads -> private case
- Call the most efficient underlying numerical libraries

General steps to achieve a good efficiency

- UPC optimization techniques:
  - Privatization of the accesses to shared memory
  - Agregation of remote shared memory accesses (`upc_memget, upc_memput, upc_memcpy`)
  - Overlapping remote accesses with computation
- Correct distribution of the workload and data among threads -> private case
- Call the most efficient underlying numerical libraries

Finis Terrae (CESGA)

142 HP Integrity rx7640 nodes, each:

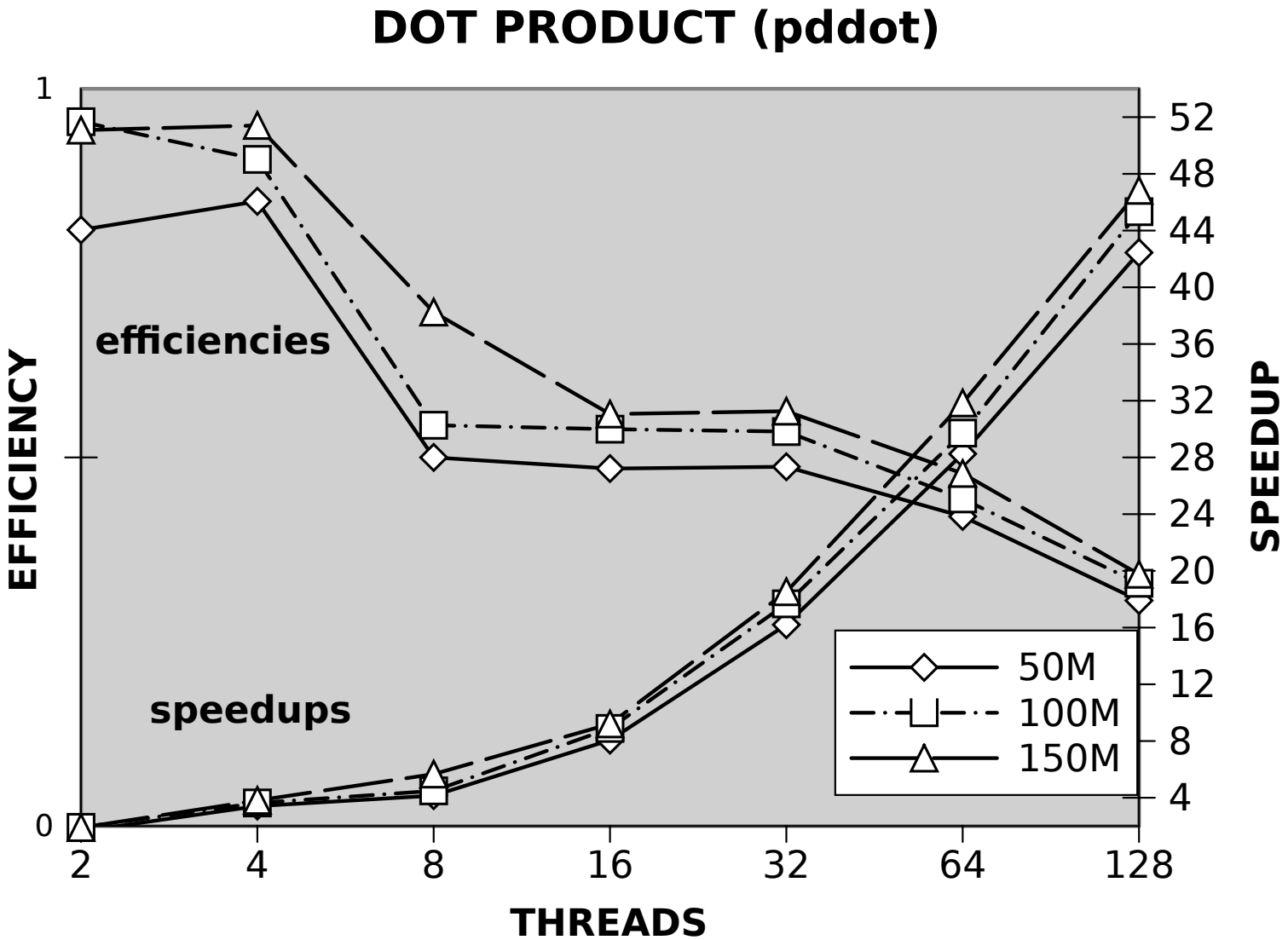
- 16 Montvale Itanium2 (IA64) cores at 1.6 GHz
  - 2 cells, each
    - 4 dual-core procesors
    - 1 shared memory module
- 128 GB RAM
- Mellanox InfiniBand HCA (16 Gbps bandwidth)

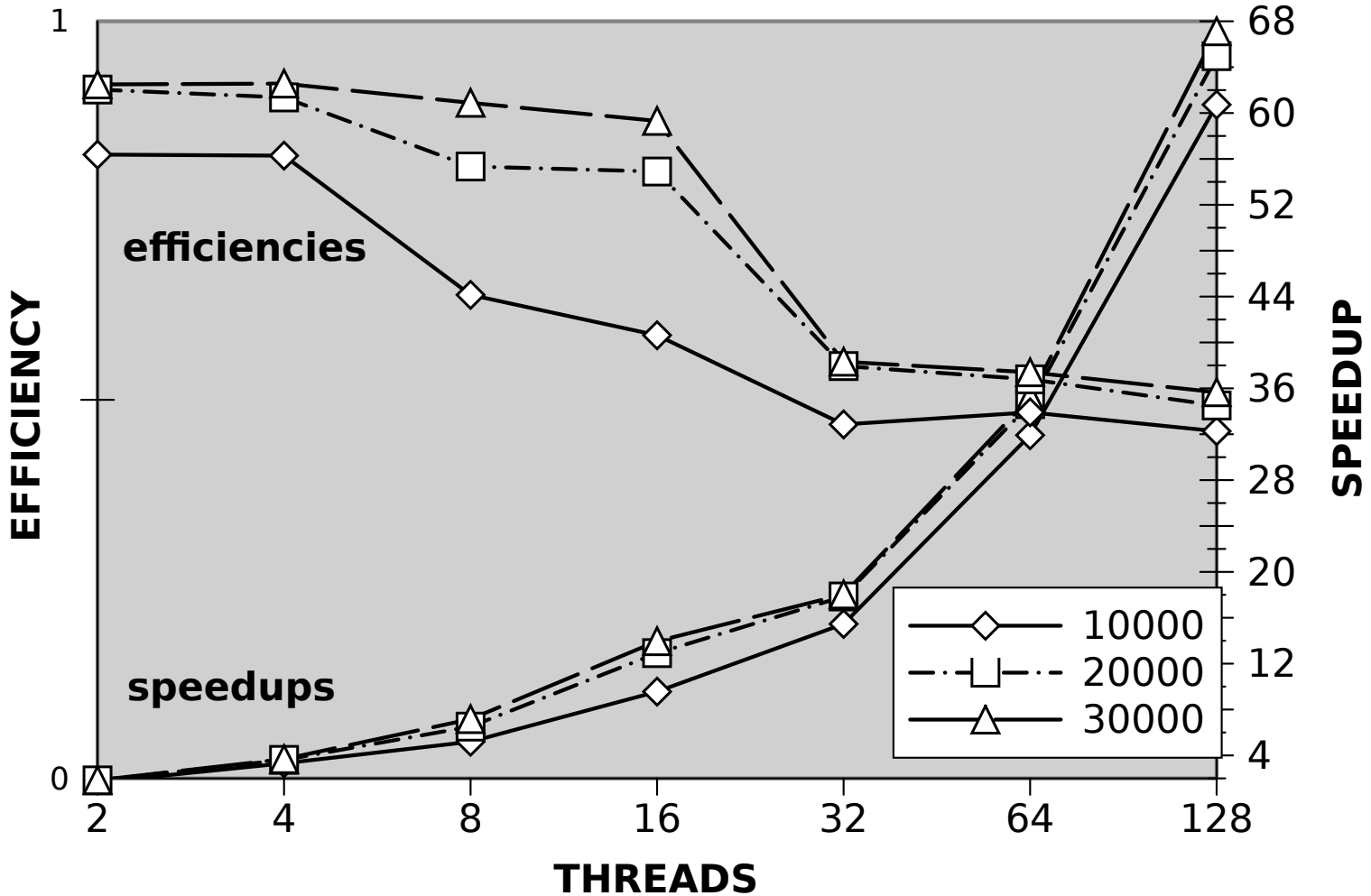
SW Configuration:

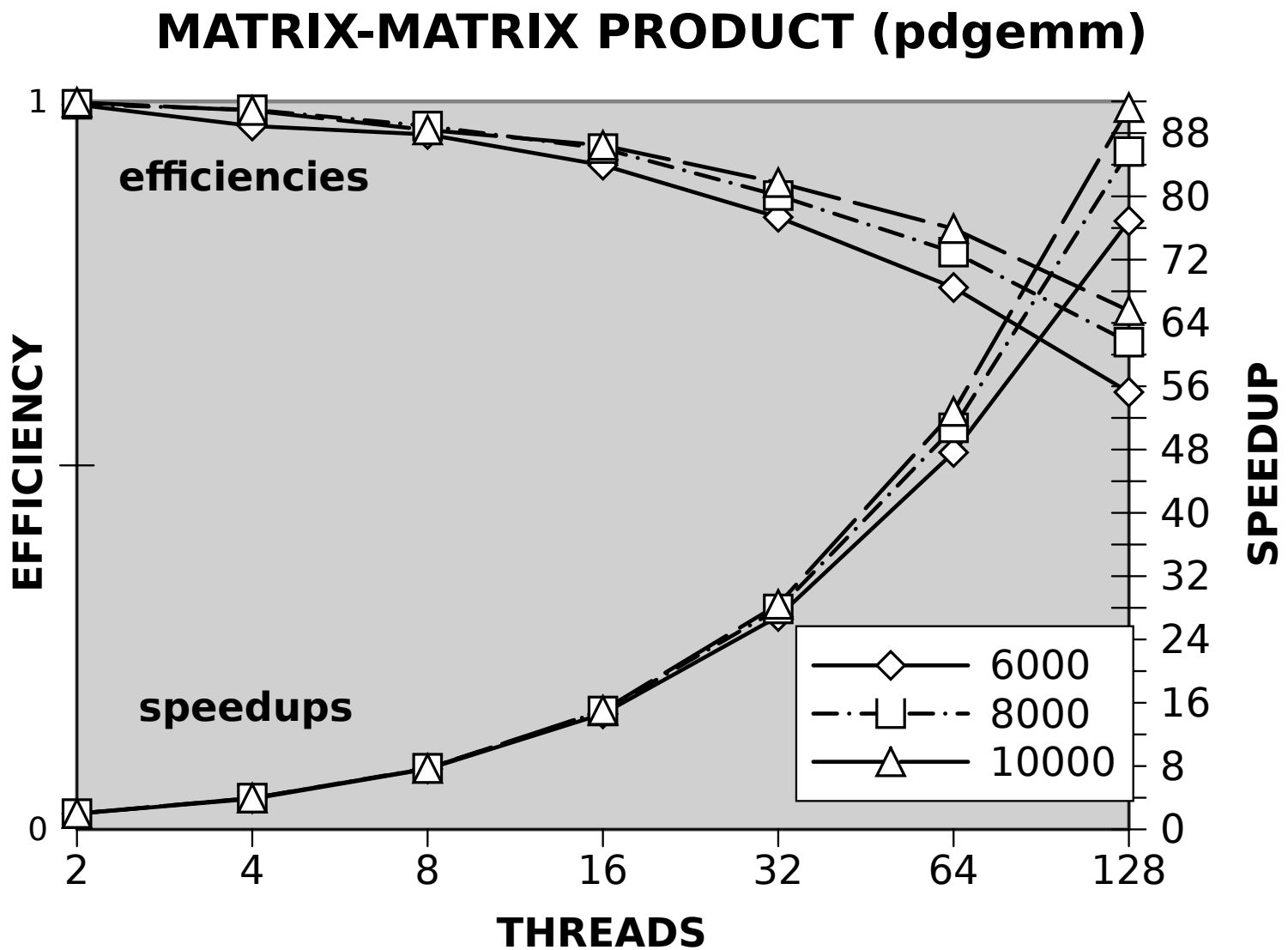
- Berkeley UPC (BUPC) 2.6
- Intel Math Kernel Library (MKL) 9.1
  - All sequential routines BLAS1, BLAS2 and BLAS3
  - Other routines: SparseBLAS, LAPACK, ScaLAPACK...

Configuration of benchmarks

- Hybrid memory configuration
  - Locality exploitation of threads in the same node -> shared memory
  - Enhancing scalability -> distributed memory
- 4 threads per node, 2 per cell
- Private version
- `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
10000
20000
30000
1
0
68
60
52
44
36
28
20
12
4
2
4
8
16
32
64
128


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

Summary

- First parallel numerical library developed for UPC -> Novelty
- Allows to store input and output data in private or shared memory -> Flexibility
- Use of the standard sequential BLAS functions -> Portability
- Scalability demonstrated by experimental tests -> Efficiency

Future work

Develop a sparse counterpart library for UPC

Summary

- First parallel numerical library developed for UPC -> Novelty
- Allows to store input and output data in private or shared memory -> Flexibility
- Use of the standard sequential BLAS functions -> Portability
- Scalability demonstrated by experimental tests -> Efficiency

Future work

Develop a sparse counterpart library for UPC

Questions?

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

Computer Architecture Group, Dept. of Electronics and Systems

University of A Coruña, Spain