# *Curso Programación en el SuperComputador BEN-ARABI*
# *Librerías de Cálculo Científico*

Luis P. García González
Universidad Politécnica de Cartagena
Servicio de Apoyo a la Investigación Tecnológica

10 de febrero de 2012

# Resumen

# Resumen

# Resumen

# Resumen

# Resumen

# Resumen

# Resumen

# Resumen

# Resumen

# Introducción

- Web del Curso:
  `http://www.ditec.um.es/~javiercm/curso_psba`
- Librerías BLAS, LAPACK y ScaLAPACK.
- Implementación de Intel MKL en el SuperComputador Ben-Arabi. Otras rutinas incluidas en la implementación MKL.
- APIs para C, FORTRAN, Compilación, Lincado y prestaciones.
- Explotación del Paralelismo con Intel MKL.

# Introducción

- Web del Curso:
  `http://www.ditec.um.es/~javiercm/curso_psba`
- Librerías BLAS, LAPACK y ScaLAPACK.
- Implementación de Intel MKL en el SuperComputador Ben-Arabi. Otras rutinas incluidas en la implementación MKL.
- APIs para C, FORTRAN, Compilación, Lincado y prestaciones.
- Explotación del Paralelismo con Intel MKL.

# Introducción

- Web del Curso:
  `http://www.ditec.um.es/~javiercm/curso_psba`
- Librerías BLAS, LAPACK y ScaLAPACK.
- Implementación de Intel MKL en el SuperComputador Ben-Arabi. Otras rutinas incluidas en la implementación MKL.
- APIs para C, FORTRAN, Compilación, Lincado y prestaciones.
- Explotación del Paralelismo con Intel MKL.

# Introducción

- Web del Curso:
  `http://www.ditec.um.es/~javiercm/curso_psba`
- Librerías BLAS, LAPACK y ScaLAPACK.
- Implementación de Intel MKL en el SuperComputador Ben-Arabi. Otras rutinas incluidas en la implementación MKL.
- APIs para C, FORTRAN, Compilación, Lincado y prestaciones.
- Explotación del Paralelismo con Intel MKL.

# Introducción

- Web del Curso:
  `http://www.ditec.um.es/~javiercm/curso_psba`
- Librerías BLAS, LAPACK y ScaLAPACK.
- Implementación de Intel MKL en el SuperComputador Ben-Arabi. Otras rutinas incluidas en la implementación MKL.
- APIs para C, FORTRAN, Compilación, Lincado y prestaciones.
- Explotación del Paralelismo con Intel MKL.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# BLAS y LAPACK

- BLAS (`http://www.netlib.org/blas`): Basic Linear Algebra Subprograms. Conjunto de rutinas estándar que proporcionan operaciones básicas con vectores y matrices. Organizada en niveles:
  - Nivel 1: Operaciones escalares, vectores, vector-vector.
  - Nivel 2: Operaciones matriz-vector.
  - Nivel 3: Operaciones matriz-matriz.
- LAPACK (`http://www.netlib.org/lapack`): Linear Algebra PACKage. Conjunto de rutinas para trabajar con matrices densas y matrices banda:
  - Resolución de sistemas de ecuaciones lineales.
  - Resolución por mínimos cuadrados de sistemas de ecuaciones lineales.
  - Resolución de problemas de autovalores y valores singulares.
  - Factorización de matrices LU, Cholesky, QR, SVD, Schur.

# ScaLAPACK

- ScaLAPACK (`http://www.netlib.org/scalapack`): Scalable LAPACK. Un subconjunto de LAPACK para sistemas paralelos de memoria distribuida/compartida. Modelo de programación paralela Single-Program-Multiple-Data y utiliza programación por paso de mensajes (MPI) para la comunicación entre procesos. Características:
  - Asume una *descomposición cíclica por bloques en dos dimensiones* para las matrices.
  - Al igual que LAPACK las rutinas están basadas en algoritmos que trabajan con bloques de la matriz.
  - Las rutinas son similares a su equivalente en LAPACK.
  - Construida a partir de una versión de BLAS para memoria distribuida (PBLAS) y de unas rutinas de comunicación específicas (BLACS).

# ScaLAPACK

- ScaLAPACK (`http://www.netlib.org/scalapack`): Scalable LAPACK. Un subconjunto de LAPACK para sistemas paralelos de memoria distribuida/compartida. Modelo de programación paralela Single-Program-Multiple-Data y utiliza programación por paso de mensajes (MPI) para la comunicación entre procesos. Características:
  - Asume una *descomposición cíclica por bloques en dos dimensiones* para las matrices.
  - Al igual que LAPACK las rutinas están basadas en algoritmos que trabajan con bloques de la matriz.
  - Las rutinas son similares a su equivalente en LAPACK.
  - Construida a partir de una versión de BLAS para memoria distribuida (PBLAS) y de unas rutinas de comunicación específicas (BLACS).

# ScaLAPACK

- ScaLAPACK (`http://www.netlib.org/scalapack`): Scalable LAPACK. Un subconjunto de LAPACK para sistemas paralelos de memoria distribuida/compartida. Modelo de programación paralela Single-Program-Multiple-Data y utiliza programación por paso de mensajes (MPI) para la comunicación entre procesos. Características:
  - Asume una *descomposición cíclica por bloques en dos dimensiones* para las matrices.
  - Al igual que LAPACK las rutinas están basadas en algoritmos que trabajan con bloques de la matriz.
  - Las rutinas son similares a su equivalente en LAPACK.
  - Construida a partir de una versión de BLAS para memoria distribuida (PBLAS) y de unas rutinas de comunicación específicas (BLACS).

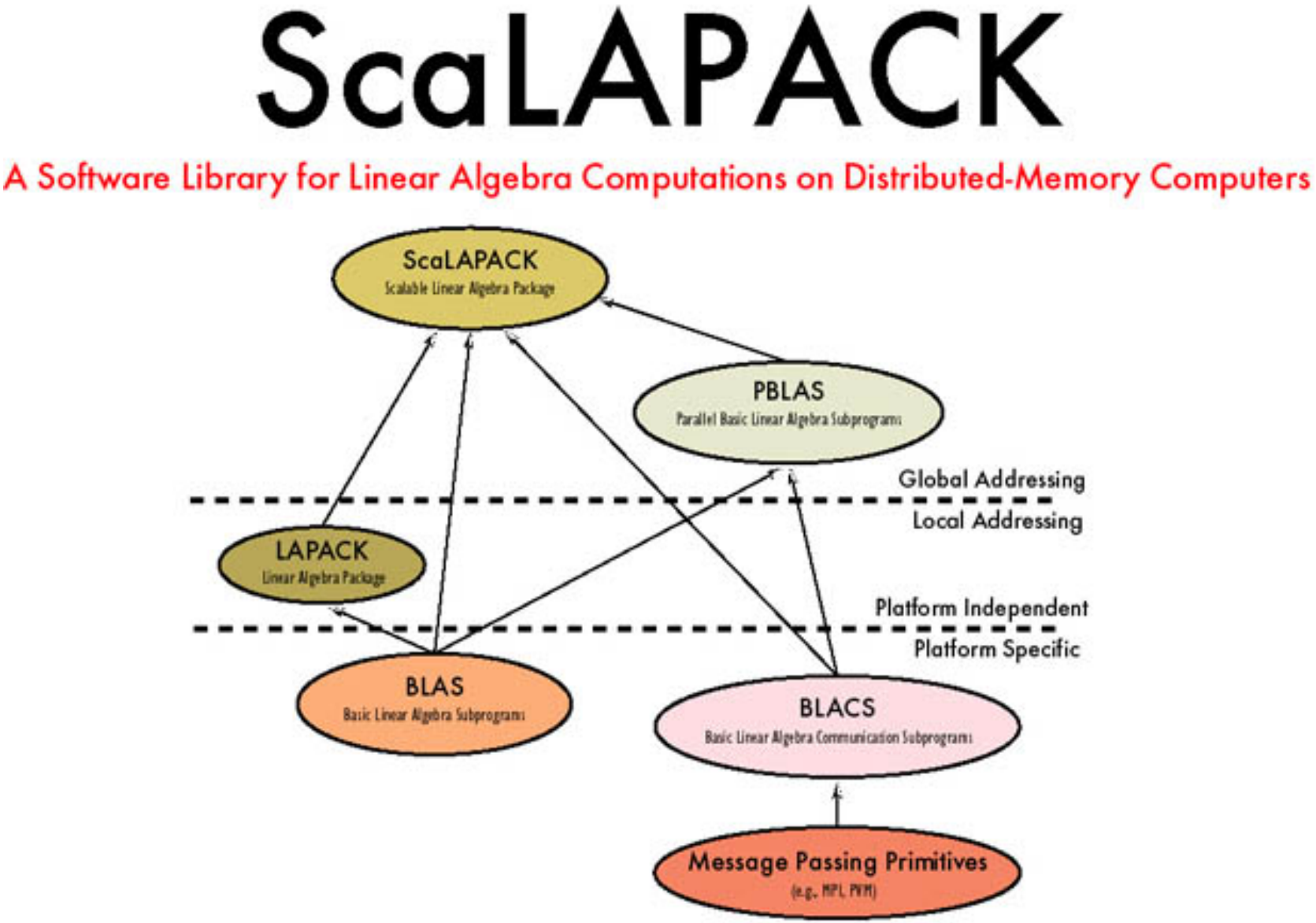
# ScaLAPACK

- ScaLAPACK (`http://www.netlib.org/scalapack`): Scalable LAPACK. Un subconjunto de LAPACK para sistemas paralelos de memoria distribuida/compartida. Modelo de programación paralela Single-Program-Multiple-Data y utiliza programación por paso de mensajes (MPI) para la comunicación entre procesos. Características:
  - Asume una *descomposición cíclica por bloques en dos dimensiones* para las matrices.
  - Al igual que LAPACK las rutinas están basadas en algoritmos que trabajan con bloques de la matriz.
  - Las rutinas son similares a su equivalente en LAPACK.
  - Construida a partir de una versión de BLAS para memoria distribuida (PBLAS) y de unas rutinas de comunicación específicas (BLACS).

# ScaLAPACK

- ScaLAPACK (`http://www.netlib.org/scalapack`): Scalable LAPACK. Un subconjunto de LAPACK para sistemas paralelos de memoria distribuida/compartida. Modelo de programación paralela Single-Program-Multiple-Data y utiliza programación por paso de mensajes (MPI) para la comunicación entre procesos. Características:
  - Asume una *descomposición cíclica por bloques en dos dimensiones* para las matrices.
  - Al igual que LAPACK las rutinas están basadas en algoritmos que trabajan con bloques de la matriz.
  - Las rutinas son similares a su equivalente en LAPACK.
  - Construida a partir de una versión de BLAS para memoria distribuida (PBLAS) y de unas rutinas de comunicación específicas (BLACS).

# Esquema ScaLAPACK



# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Contenidos de MKL

- Versión optimizada de BLAS para procesadores Intel.
- Sparse BLAS. BLAS para vectores/matrices con estructura dispersa.
- LAPACK y ScaLAPACK.
- DFT. Version de la Transformada Rápida de Fourier.
- Sparse Solvers (PARDISO, DSS y ISS). Resolución de sistemas de ecuaciones lineales.
- VML. Vector Math Library. Conjunto de funciones transcendentes vectorizadas. Similar a la librería libm pero más rápida.
- VSL. Vector Statistical Library. Conjunto de rutinas de generación de números aleatorios.
- PDEs y Optimization Solvers
- GMP. Rutinas para operaciones aritméticas con enteros de precisión arbitraria.

# Entorno de trabajo de MKL

## Tipos de datos soportados

- Precisión Simple y Doble para números reales y complejos.

## API C/C++, Fortran y Java.

| Rutinas | Fortran 77 | Fortran 95/99 | C/C++ |
|---|---|---|---|
| BLAS | * | * | Via CBLAS |
| Sparse BLAS | * | * | * |
| LAPACK | * | * | |
| ScaLAPACK | * | | |
| PARDISO | * | | * |
| DSS/ISS | * | * | * |
| VML/VSL | | * | * |
| FFT/Cluster FFT | | * | * |
| PDEs | | * | * |
| Optimization Solver | * | * | * |

# Entorno de trabajo de MKL

## Tipos de datos soportados

- Precisión Simple y Doble para números reales y complejos.

## API C/C++, Fortran y Java.

| Rutinas | Fortran 77 | Fortran 95/99 | C/C++ |
|---|---|---|---|
| BLAS | * | * | Via CBLAS |
| Sparse BLAS | * | * | * |
| LAPACK | * | * | |
| ScaLAPACK | * | | |
| PARDISO | * | | * |
| DSS/ISS | * | * | * |
| VML/VSL | | * | * |
| FFT/Cluster FFT | | * | * |
| PDEs | | * | * |
| Optimization Solver | * | * | * |

# Convenciones Rutinas BLAS

## BLAS utiliza un esquema para dar nombre a las rutinas

`<character>` `<name>` `<mod>` `()`

## El campo `<character>` indica el tipo de dato con el que trabaja:

| | |
|---|---|
| `s` | real, precisión simple |
| `c` | complejo, precisión simple |
| `d` | real, precisión doble |
| `z` | complejo, precisión doble |

Algunas rutinas pueden tener estos códigos combinados, y aparecer `sc` o `dz`

# Convenciones Rutinas BLAS

## Campo `<name>`

En BLAS nivel 2 y 3, indica el tipo de matriz (Hay 11 tipos distintos):

| | |
|---|---|
| `ge` | matriz general |
| `gb` | matriz banda general |
| `sy` | matriz simétrica |
| `sb` | matriz banda simétrica |
| `tr` | matriz triangular |
| `tp` | matriz triangular empaquetada |
| `he` | matriz Hermitiana |

En BLAS nivel 1, indican el tipo de operación (`?rot, ?swap`).

# Convenciones Rutinas BLAS

**Campo <mod>, si presente, informa sobre detalles adicionales de la operación**

| | En BLAS nivel 2: | | En BLAS nivel 3: |
|---|---|---|---|
| `mv` | producto matriz-vector | `mm` | producto matriz-matriz |
| `r` | actualización rango-1 | `rk` | actualización rango-k |
| `r2` | actualización rango-2 | `r2k` | actualización rango-2k |

- `dgemm` producto matriz-matriz, matriz general, real, doble precisión
- `cgemv` producto matriz-vector, matriz general, complejo, precisión simple
- `ddot` producto vectores, real, doble precisión

# Convenciones Rutinas BLAS

*Campo <mod>, si presente, informa sobre detalles adicionales de la operación*

| | En BLAS nivel 2: | | En BLAS nivel 3: |
|---|---|---|---|
| `mv` | producto matriz-vector | `mm` | producto matriz-matriz |
| `r` | actualización rango-1 | `rk` | actualización rango-k |
| `r2` | actualización rango-2 | `r2k` | actualización rango-2k |

- `dgemm` producto matriz-matriz, matriz general, real, doble precisión
- `cgemv` producto matriz-vector, matriz general, complejo, precisión simple
- `ddot` producto vectores, real, doble precisión

# Convenciones Rutinas BLAS

**Campo <mod>, si presente, informa sobre detalles adicionales de la operación**

| | En BLAS nivel 2: | | En BLAS nivel 3: |
|---|---|---|---|
| `mv` | producto matriz-vector | `mm` | producto matriz-matriz |
| `r` | actualización rango-1 | `rk` | actualización rango-k |
| `r2` | actualización rango-2 | `r2k` | actualización rango-2k |

- `dgemm` producto matriz-matriz, matriz general, real, doble precisión
- `cgemv` producto matriz-vector, matriz general, complejo, precisión simple
- `ddot` producto vectores, real, doble precisión

# Convenciones Rutinas BLAS

**Campo <mod>, si presente, informa sobre detalles adicionales de la operación**

| | En BLAS nivel 2: | | En BLAS nivel 3: |
|---|---|---|---|
| `mv` | producto matriz-vector | `mm` | producto matriz-matriz |
| `r` | actualización rango-1 | `rk` | actualización rango-k |
| `r2` | actualización rango-2 | `r2k` | actualización rango-2k |

- `dgemm` producto matriz-matriz, matriz general, real, doble precisión
- `cgemv` producto matriz-vector, matriz general, complejo, precisión simple
- `ddot` producto vectores, real, doble precisión

# Interface MKL BLAS

**Description**

The `?dot` routines perform a vector-vector reduction operation defined as $res = \sum((x * y))$, where `x` and `y` are vectors.

```
res = sdot(n, x, incx, y, incy)
res = sdot(n, x, incx, y, incy)
```

**Input Parametres**

| | |
|---|---|
| `n` | `INTEGER`. Specifies the number of elements in vectors `x` and `y` |
| `x` | `REAL` for `sdot`<br>`DOUBLE PRECISION` for `ddot`<br>Array, `DIMENSION` at least `(1+(n-1)*abs(incx))` |
| `incx` | `INTEGER`. Specifies the increment for the elements of `x` |
| `y` | `REAL` for `sdot`<br>`DOUBLE PRECISION` for `ddot`<br>Array, `DIMENSION` at least `(1+(n-1)*abs(incy))` |
| `incy` | `INTEGER`. Specifies the increment for the elements of y. |

**Output Parametres**

| | |
|---|---|
| `res` | `REAL` for `sdot`<br>`DOUBLE PRECISION` for `ddot`<br>Contains the result of the dot product of `x` and `y`, if `n` is positive. Otherwise, `res` contains `0`. |

# Llamada en FORTRAN a rutinas BLAS

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n=5
incx = 2
incy = 1
do i = 1, 10
   x(i) = 2.0e0
   y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

# Llamada en FORTRAN a rutinas BLAS

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n=5
incx = 2
incy = 1
do i = 1, 10
   x(i) = 2.0e0
   y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

# Llamada en FORTRAN a rutinas BLAS

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n=5
incx = 2
incy = 1
do i = 1, 10
   x(i) = 2.0e0
   y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

# Llamada en FORTRAN a rutinas BLAS

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n=5
incx = 2
incy = 1
do i = 1, 10
   x(i) = 2.0e0
   y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

# Llamada en FORTRAN a rutinas BLAS

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n=5
incx = 2
incy = 1
do i = 1, 10
   x(i) = 2.0e0
   y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

# Llamada en FORTRAN a rutinas BLAS

```
program dot_main
real x(10), y(10), sdot, res
integer n, incx, incy, i
external sdot
n=5
incx = 2
incy = 1
do i = 1, 10
   x(i) = 2.0e0
   y(i) = 1.0e0
end do
res = sdot (n, x, incx, y, incy)
print*, 'SDOT = ', res
end
```

# Llamada en C a rutinas BLAS

```
extern float sdot_(int *, float *, int *,
                   float *, int *);
int main(void) {
  float x[10], y[10], res;
  int n, incx, incy, i;
  n = 5;
  incx = 2;
  incy = 1;
  for (i=0; i < 9; i++) {
    x[i] = 2.0;
    y[i] = 1.0;
  }
  res = sdot_(&n, x, &incx, y, &incy);
  printf("SDOT =%f\n", res);
  return 0;
}
```

# Llamada en C a rutinas BLAS

```
extern float sdot_(int *, float *, int *,
                   float *, int *);
int main(void) {
  float x[10], y[10], res;
  int n, incx, incy, i;
  n = 5;
  incx = 2;
  incy = 1;
  for (i=0; i < 9; i++) {
    x[i] = 2.0;
    y[i] = 1.0;
 }
  res = sdot_(&n, x, &incx, y, &incy);
  printf("SDOT =%f\n", res);
  return 0;
}
```

# Llamada en C a rutinas BLAS

```
extern float sdot_(int *, float *, int *,
                   float *, int *);
int main(void) {
  float x[10], y[10], res;
  int n, incx, incy, i;
  n = 5;
  incx = 2;
  incy = 1;
  for (i=0; i < 9; i++) {
    x[i] = 2.0;
    y[i] = 1.0;
  }
  res = sdot_(&n, x, &incx, y, &incy);
  printf("SDOT =%f\n", res);
  return 0;
}
```

# Llamada en C a rutinas BLAS

```
extern float sdot_(int *, float *, int *,
                   float *, int *);
int main(void) {
  float x[10], y[10], res;
  int n, incx, incy, i;
  n = 5;
  incx = 2;
  incy = 1;
  for (i=0; i < 9; i++) {
    x[i] = 2.0;
    y[i] = 1.0;
 }
  res = sdot_(&n, x, &incx, y, &incy);
  printf("SDOT =%f\n", res);
  return 0;
}
```

# Llamada en C a rutinas BLAS

```
extern float sdot_(int *, float *, int *,
                   float *, int *);
int main(void) {
  float x[10], y[10], res;
  int n, incx, incy, i;
  n = 5;
  incx = 2;
  incy = 1;
  for (i=0; i < 9; i++) {
    x[i] = 2.0;
    y[i] = 1.0;
 }
  res = sdot_(&n, x, &incx, y, &incy);
  printf("SDOT =%f\n", res);
  return 0;
}
```

# Llamada en C a rutinas BLAS

```
extern float sdot_(int *, float *, int *,
                   float *, int *);
int main(void) {
  float x[10], y[10], res;
  int n, incx, incy, i;
  n = 5;
  incx = 2;
  incy = 1;
  for (i=0; i < 9; i++) {
    x[i] = 2.0;
    y[i] = 1.0;
 }
  res = sdot_(&n, x, &incx, y, &incy);
  printf("SDOT =%f\n", res);
  return 0;
}
```

# Llamada en C a rutinas CBLAS

```
#include "mkl_types.h"

#include "mkl_cblas.h"

MKL_INT    n, incx, incy, i;

res = cblas_sdot(n, x, incx, y, incy);
```

# Llamada en C a rutinas CBLAS

```
#include "mkl_types.h"

#include "mkl_cblas.h"

MKL_INT    n, incx, incy, i;

res = cblas_sdot(n, x, incx, y, incy);
```

# Llamada en C a rutinas CBLAS

```
#include "mkl_types.h"

#include "mkl_cblas.h"

MKL_INT    n, incx, incy, i;

res = cblas_sdot(n, x, incx, y, incy);
```

# Convenciones Rutinas LAPACK

## *LAPACK utiliza un esquema para dar nombre a las rutinas*

`?yyzzz` (Para rutinas computacionales) o `?yyzz` (Para rutinas "driver")

## *El campo ? indica el tipo de dato con el que trabaja:*

| | |
|---|---|
| `s` | real, precisión simple |
| `c` | complejo, precisión simple |
| `d` | real, precisión doble |
| `z` | complejo, precisión doble |

Algunas rutinas pueden tener estos códigos combinados, y aparecer `ds` o `zc`

# Convenciones Rutinas LAPACK

## Campo `yy`

Indica el tipo de matriz y el esquema de almacenamiento Hay 15 tipos distintos:

| | |
|---|---|
| `ge` | matriz general |
| `gb` | matriz banda general |
| `sy` | matriz simétrica indefinida |
| `po` | matriz simétrica o Hermitiana definida positiva |
| `tr` | matriz triangular |
| `tp` | matriz triangular empaquetada |
| `he` | matriz Hermitiana indefinida |

# Convenciones Rutinas LAPACK

**Campo `zzz`, en rutinas computacionales, indica el tipo de operación realizada**

| | |
|---|---|
| `trf` | factorización triangular de la matriz |
| `trs` | resuelve el sistema lineal dada la matriz factorizada |
| `tri` | obtiene la matriz inversa a partir de la matriz factorizada |

**Campo `zz`, en rutinas "driver"**

| | |
|---|---|
| `?sv` | rutina simple |
| `?svx` | rutina que trabaja en modo experto |
| `?svxx` | modo experto, con precisión extra y refinamiento iterativo |

# Convenciones Rutinas LAPACK

## *Campo `zzz`, en rutinas computacionales, indica el tipo de operación realizada*

| | |
|---|---|
| `trf` | factorización triangular de la matriz |
| `trs` | resuelve el sistema lineal dada la matriz factorizada |
| `tri` | obtiene la matriz inversa a partir de la matriz factorizada |

## *Campo `zz`, en rutinas "driver"*

| | |
|---|---|
| `?sv` | rutina simple |
| `?svx` | rutina que trabaja en modo experto |
| `?svxx` | modo experto, con precisión extra y refinamiento iterativo |

# Interface MKL LAPACK

**Description**

The `?getrf` routines computes the *LU* factorization of a general `m`-by-`n` matrix *A* as $A = P * L * U$, where *P* is a permutation matrix.

**Input Parametres**

| | |
|---|---|
| `m` | `INTEGER`. Specifies the number of rows in the matrix *A* ($m \geqslant 0$) |
| `n` | `INTEGER`. Specifies the number of columns in *A* ($n \geqslant 0$) |
| `a` | `REAL` for `sgetrf`<br>`DOUBLE PRECISION` for `dgetrf`<br>Array, `DIMENSION (lda,*)`. Contain the matrix *A*. The second dimension of `a` must be at least `max(1,n)`. |
| `lda` | `INTEGER`. The first dimension of array `a`. |

**Output Parametres**

| | |
|---|---|
| `a` | Overwritten by *L* and *U*. The unit diagonal elements of *L* are not stored. |
| `ipiv` | `INTEGER`<br>Array, `DIMENSION` at least `max(1,min(m,n))`. The pivot indices for $1 \leqslant i \leqslant \min(m, n)$, row `i` was interchanged with row `ipiv(i)`. |
| `info` | `INTEGER`. If `info = 0`, the execution is successful.<br>If `info = -i`, the `i`-th parameter had an ilegal value. |

# Llamada en FORTRAN a rutinas LAPACK

```
*     .. Escalares Locales ..
      INTEGER           I, INFO, J, M, N
*     .. Arrays Locales ..
      DOUBLE PRECISION A(LDA,NMAX)
      INTEGER           IPIV(NMAX)
*     .. Subrutinas Externas ..
      EXTERNAL          DGETRF
*
*     Factoriza A
*
      CALL DGETRF(M,N,A,LDA,IPIV,INFO)
*
*     Imprimir indices pivote
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'IPIV'
      WRITE (NOUT,99999) (IPIV(I),I=1,MIN(M,N))
*
      IF (INFO.NE.0) WRITE (NOUT,*) 'El factor U es singular'
*
      END IF
      STOP
* 99999 FORMAT ((3X,7I11))
      END
```

# Llamada en C a rutinas LAPACK

```
extern void dgetrf_( int *, int *, double *, int *, int *, int *);

#define max_size  10

int main() {
    double *a;
    int ipvt[max_size];
    int m, lda, info;
    int status;
    int i;

    a = (double *)malloc(max_size*max_size*sizeof(double));

    dgetrf_(&m, &m, a, &lda, ipvt, &info);
    printf ("\n  info despues de dgetrf %d\n", info);

    printf("  Vector Pivote: \n");
    for (i = 0; i < m; i++)
       printf ("    %d\n", ipvt[i]);

    free (a);

}  /* fin de main() */
```

# Mezclando lenguajes de programación con Intel MKL

Dado que las rutinas de BLAS y LAPACK están escritas en FORTRAN, cuando se utilizan desde un programa escrito en el lenguaje de programación C, hay que tener en cuenta las siguientes consideraciones:

- Las variables se pasan a las rutinas por dirección NO POR valor
- Se almacenan los datos al estilo FORTRAN, esto es, en orden "column-major" en lugar de en orden "row-major".
- Por ejemplo, si una matriz $A$ de tamaño $m \times n$ es almacenada por columnas en un array de una dimensión $B$, la forma de acceder sería:
- $A[i][j] = B[i + j * m]$ en C $(i = 0, \ldots, m-1; j = 0, \ldots, n-1)$
- $A(i, j) = B(i + j * m)$ en FORTRAN $(i = 1, \ldots, m; j = 1, \ldots, n)$

# Mezclando lenguajes de programación con Intel MKL

Dado que las rutinas de BLAS y LAPACK están escritas en FORTRAN, cuando se utilizan desde un programa escrito en el lenguaje de programación C, hay que tener en cuenta las siguientes consideraciones:

- Las variables se pasan a las rutinas por dirección NO POR valor
- Se almacenan los datos al estilo FORTRAN, esto es, en orden "column-major" en lugar de en orden "row-major".
- Por ejemplo, si una matriz $A$ de tamaño $m \times n$ es almacenada por columnas en un array de una dimensión $B$, la forma de acceder sería:
- $A[i][j] = B[i + j * m]$ en C $(i = 0, \ldots, m-1; j = 0, \ldots, n-1)$
- $A(i, j) = B(i + j * m)$ en FORTRAN $(i = 1, \ldots, m; j = 1, \ldots, n)$

# Mezclando lenguajes de programación con Intel MKL

Dado que las rutinas de BLAS y LAPACK están escritas en FORTRAN, cuando se utilizan desde un programa escrito en el lenguaje de programación C, hay que tener en cuenta las siguientes consideraciones:

- Las variables se pasan a las rutinas por dirección NO POR valor
- Se almacenan los datos al estilo FORTRAN, esto es, en orden "column-major" en lugar de en orden "row-major".
- Por ejemplo, si una matriz $A$ de tamaño $m \times n$ es almacenada por columnas en un array de una dimensión $B$, la forma de acceder sería:
- $A[i][j] = B[i + j * m]$ en C $(i = 0, \ldots, m-1; j = 0, \ldots, n-1)$
- $A(i, j) = B(i + j * m)$ en FORTRAN $(i = 1, \ldots, m; j = 1, \ldots, n)$

# Mezclando lenguajes de programación con Intel MKL

Dado que las rutinas de BLAS y LAPACK están escritas en FORTRAN, cuando se utilizan desde un programa escrito en el lenguaje de programación C, hay que tener en cuenta las siguientes consideraciones:

- Las variables se pasan a las rutinas por dirección NO POR valor
- Se almacenan los datos al estilo FORTRAN, esto es, en orden "column-major" en lugar de en orden "row-major".
- Por ejemplo, si una matriz $A$ de tamaño $m \times n$ es almacenada por columnas en un array de una dimensión $B$, la forma de acceder sería:
- $A[i][j] = B[i + j * m]$ en C $(i = 0, \ldots, m-1; j = 0, \ldots, n-1)$
- $A(i, j) = B(i + j * m)$ en FORTRAN $(i = 1, \ldots, m; j = 1, \ldots, n)$

# Mezclando lenguajes de programación con Intel MKL

Dado que las rutinas de BLAS y LAPACK están escritas en FORTRAN, cuando se utilizan desde un programa escrito en el lenguaje de programación C, hay que tener en cuenta las siguientes consideraciones:

- Las variables se pasan a las rutinas por dirección NO POR valor
- Se almacenan los datos al estilo FORTRAN, esto es, en orden "column-major" en lugar de en orden "row-major".
- Por ejemplo, si una matriz $A$ de tamaño $m \times n$ es almacenada por columnas en un array de una dimensión $B$, la forma de acceder sería:
- $A[i][j] = B[i + j * m]$ en C $(i = 0, \ldots, m-1; j = 0, \ldots, n-1)$
- $A(i, j) = B(i + j * m)$ en FORTRAN $(i = 1, \ldots, m; j = 1, \ldots, n)$

# Mezclando lenguajes de programación con Intel MKL

Dado que las rutinas de BLAS y LAPACK están escritas en FORTRAN, cuando se utilizan desde un programa escrito en el lenguaje de programación C, hay que tener en cuenta las siguientes consideraciones:

- Las variables se pasan a las rutinas por dirección NO POR valor
- Se almacenan los datos al estilo FORTRAN, esto es, en orden "column-major" en lugar de en orden "row-major".
- Por ejemplo, si una matriz $A$ de tamaño $m \times n$ es almacenada por columnas en un array de una dimensión $B$, la forma de acceder sería:
- $A[i][j] = B[i + j * m]$ en C $(i = 0, \ldots, m-1; j = 0, \ldots, n-1)$
- $A(i, j) = B(i + j * m)$ en FORTRAN $(i = 1, \ldots, m; j = 1, \ldots, n)$

# Rutinas ScaLAPACK

- Versión de LAPACK para arquitecturas de memoria compartida.
- Mismos nombres de la rutinas que LAPACK, añadiendo el prefijo p: `PDGETRF`
- El modelo para el entorno de computación está basado en una malla de procesos de dos dimensiones. Antes de cualquier llamada a rutinas de ScaLAPACK, la matriz tiene que ser distribuida en esta malla de procesos. Comunicaciones de procesos con BLACS.
- ScaLAPACK utiliza una distribución de la matriz cíclica y por bloques en dos dimensiones.
  - Balanceo del trabajo entre todos los procesos envueltos en la computación.
  - Usar BLAS nivel 3 para los cálculos locales, y asegurar rendimiento óptimo.

# Rutinas ScaLAPACK

- Versión de LAPACK para arquitecturas de memoria compartida.
- Mismos nombres de la rutinas que LAPACK, añadiendo el prefijo p: `PDGETRF`
- El modelo para el entorno de computación está basado en una malla de procesos de dos dimensiones. Antes de cualquier llamada a rutinas de ScaLAPACK, la matriz tiene que ser distribuida en esta malla de procesos. Comunicaciones de procesos con BLACS.
- ScaLAPACK utiliza una distribución de la matriz cíclica y por bloques en dos dimensiones.
  - Balanceo del trabajo entre todos los procesos envueltos en la computación.
  - Usar BLAS nivel 3 para los cálculos locales, y asegurar rendimiento óptimo.

# Rutinas ScaLAPACK

- Versión de LAPACK para arquitecturas de memoria compartida.
- Mismos nombres de la rutinas que LAPACK, añadiendo el prefijo p: `PDGETRF`
- El modelo para el entorno de computación está basado en una malla de procesos de dos dimensiones. Antes de cualquier llamada a rutinas de ScaLAPACK, la matriz tiene que ser distribuida en esta malla de procesos. Comunicaciones de procesos con BLACS.
- ScaLAPACK utiliza una distribución de la matriz cíclica y por bloques en dos dimensiones.
  - Balanceo del trabajo entre todos los procesos envueltos en la computación.
  - Usar BLAS nivel 3 para los cálculos locales, y asegurar rendimiento óptimo.

# Rutinas ScaLAPACK

- Versión de LAPACK para arquitecturas de memoria compartida.
- Mismos nombres de la rutinas que LAPACK, añadiendo el prefijo p: `PDGETRF`
- El modelo para el entorno de computación está basado en una malla de procesos de dos dimensiones. Antes de cualquier llamada a rutinas de ScaLAPACK, la matriz tiene que ser distribuida en esta malla de procesos. Comunicaciones de procesos con BLACS.
- ScaLAPACK utiliza una distribución de la matriz cíclica y por bloques en dos dimensiones.
  - Balanceo del trabajo entre todos los procesos envueltos en la computación.
  - Usar BLAS nivel 3 para los cálculos locales, y asegurar rendimiento óptimo.

# Rutinas ScaLAPACK

- Versión de LAPACK para arquitecturas de memoria compartida.
- Mismos nombres de la rutinas que LAPACK, añadiendo el prefijo p: `PDGETRF`
- El modelo para el entorno de computación está basado en una malla de procesos de dos dimensiones. Antes de cualquier llamada a rutinas de ScaLAPACK, la matriz tiene que ser distribuida en esta malla de procesos. Comunicaciones de procesos con BLACS.
- ScaLAPACK utiliza una distribución de la matriz cíclica y por bloques en dos dimensiones.
  - Balanceo del trabajo entre todos los procesos envueltos en la computación.
  - Usar BLAS nivel 3 para los cálculos locales, y asegurar rendimiento óptimo.

# Rutinas ScaLAPACK

- Versión de LAPACK para arquitecturas de memoria compartida.
- Mismos nombres de la rutinas que LAPACK, añadiendo el prefijo p: `PDGETRF`
- El modelo para el entorno de computación está basado en una malla de procesos de dos dimensiones. Antes de cualquier llamada a rutinas de ScaLAPACK, la matriz tiene que ser distribuida en esta malla de procesos. Comunicaciones de procesos con BLACS.
- ScaLAPACK utiliza una distribución de la matriz cíclica y por bloques en dos dimensiones.
  - Balanceo del trabajo entre todos los procesos envueltos en la computación.
  - Usar BLAS nivel 3 para los cálculos locales, y asegurar rendimiento óptimo.

# Rutinas ScaLAPACK. Distribución cíclica por bloques

Distribución cíclica en bloques con $\frac{n}{b} = 6$ y $p = 2 \times 3$. Los números a la izquierda y arriba de la matriz representan coordenadas de los procesos en la cuadrícula $2 \times 3$. 6 procesos mapeados en una malla de $2 \times 3$ procesos.

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 3 | 4 | 5 |

| | 0 | 1 | 2 | 0 | 1 | 2 |
|---|---|---|---|---|---|---|
| 0 | $A_{11}$ | $A_{12}$ | $A_{13}$ | $A_{14}$ | $A_{15}$ | $A_{16}$ |
| 1 | $A_{21}$ | $A_{22}$ | $A_{23}$ | $A_{24}$ | $A_{25}$ | $A_{26}$ |
| 0 | $A_{31}$ | $A_{32}$ | $A_{33}$ | $A_{34}$ | $A_{35}$ | $A_{36}$ |
| 1 | $A_{41}$ | $A_{42}$ | $A_{43}$ | $A_{44}$ | $A_{45}$ | $A_{46}$ |
| 0 | $A_{51}$ | $A_{52}$ | $A_{53}$ | $A_{54}$ | $A_{55}$ | $A_{56}$ |
| 1 | $A_{61}$ | $A_{62}$ | $A_{63}$ | $A_{64}$ | $A_{65}$ | $A_{66}$ |

| | 0 | | 1 | | 2 | |
|---|---|---|---|---|---|---|
| | $A_{11}$ | $A_{14}$ | $A_{12}$ | $A_{15}$ | $A_{13}$ | $A_{16}$ |
| 0 | $A_{31}$ | $A_{34}$ | $A_{32}$ | $A_{35}$ | $A_{33}$ | $A_{36}$ |
| | $A_{51}$ | $A_{54}$ | $A_{52}$ | $A_{55}$ | $A_{53}$ | $A_{56}$ |
| | $A_{21}$ | $A_{24}$ | $A_{22}$ | $A_{25}$ | $A_{23}$ | $A_{26}$ |
| 1 | $A_{41}$ | $A_{44}$ | $A_{42}$ | $A_{45}$ | $A_{43}$ | $A_{46}$ |
| | $A_{61}$ | $A_{64}$ | $A_{62}$ | $A_{65}$ | $A_{63}$ | $A_{66}$ |

# Rutinas ScaLAPACK. Descriptor de Arrays

Con cada array global, se asocia un *ARRAY DESCRIPTOR* que contiene información sobre la distribución de datos requerida para poder realizar el mapeo entre el array global y el correspondiente proceso y localización en memoria.

| Array Element | Name | Definition |
|---|---|---|
| 1 | dtype | Descriptor type ( =1 for dense matrices) |
| 2 | ctxt | BLACS context handle for the process grid |
| 3 | m | Number of rows in the global array |
| 4 | n | Number of columns in the global array |
| 5 | mb | Row blocking factor |
| 6 | nb | Column blocking factor |
| 7 | rsrc | Process row over which the first row of the global array is distributed |
| 8 | csrc | Process column over which the first column of the global array is distributed |
| 9 | lld | Leading dimension of the local array |

# Rutinas ScaLAPACK. Descriptor de Arrays

Con cada array global, se asocia un *ARRAY DESCRIPTOR* que contiene información sobre la distribución de datos requerida para poder realizar el mapeo entre el array global y el correspondiente proceso y localización en memoria.

| Array Element | Name | Definition |
|---|---|---|
| 1 | dtype | Descriptor type ( =1 for dense matrices) |
| 2 | ctxt | BLACS context handle for the process grid |
| 3 | m | Number of rows in the global array |
| 4 | n | Number of columns in the global array |
| 5 | mb | Row blocking factor |
| 6 | nb | Column blocking factor |
| 7 | rsrc | Process row over which the first row of the global array is distributed |
| 8 | csrc | Process column over which the first column of the global array is distributed |
| 9 | lld | Leading dimension of the local array |

# Interface MKL ScaLAPACK

**Description**

The `p?getrf` routines computes the *LU* factorization of a general `m`-by-`n` distributed matrix *A* as $A = P * L * U$, where *P* is a permutation matrix.

**Input Parametres**

| | |
|---|---|
| `m` | (global) `INTEGER`. Specifies the number of rows in the matrix *A* ($m \geqslant 0$) |
| `n` | (global) `INTEGER`. Specifies the number of columns in *A* ($n \geqslant 0$) |
| `a` | (local)<br>Pointer into the local memory to an array de local dimension<br>Contains the local pieces of the distributed matrix *A*<br>to be factored. |
| `ia,ja` | (global) `INTEGER`. The row and column indices in the global<br>array *A* indicating the first row and the first column. |
| `desca` | (local and global) `INTEGER` array, dimension `dlen_`.<br>The array descriptor for the distributed matrix *A*. |

**Output Parametres**

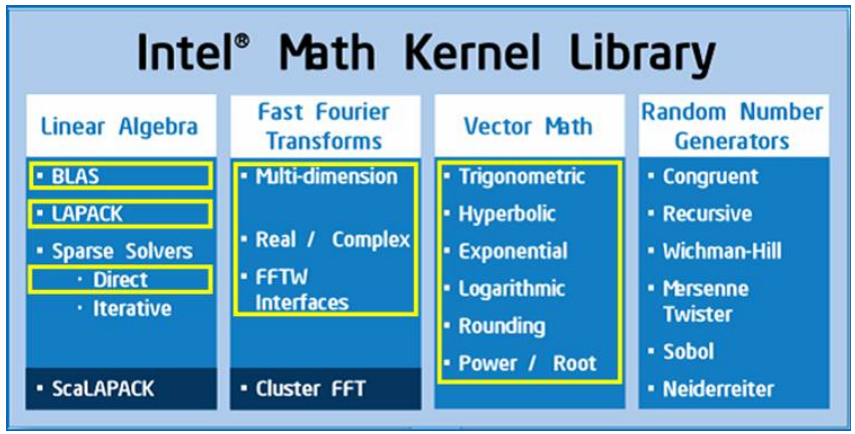
| | |
|---|---|
| `a` | Overwritten by local pieces of the factors *L* and *U*.<br>The unit diagonal elements of *L* are not stored. |
| `ipiv` | (local) `INTEGER` array<br>This array contains the pivoting information. |
| `info` | (global) `INTEGER`. If `info = 0`, the execution is successful.<br>If `info = -i`, the `i`-th parameter had an ilegal value. |

# Llamada en C a rutinas ScaLAPACK

```
extern void pdgetrf_(int *, int *, double *, int *, int *, int *,
                     int *, int *);
void main(int argc, char*argv[]) {
  double *a;
  double *local_a;
  int contxt;
  int DESCA[9];

  MPI_Init(&argc, &argv);
  Cblacs_pinfo(&myrank, &nprocs);
  Cblacs_get(0, 0, &contxt);
  Cblacs_gridinit(&contxt, "R", nprows, npcols);
  Cblacs_gridinfo(contxt, &nprows, &npcols, &myprow, &mypcol);

  local_rows_a = numroc_(&m, &mb, &myprow, &i_zero, &nprows);
  local_cols_a = numroc_(&n, &nb, &mypcol, &i_zero, &npcols);

  descinit_(DESCA, &m, &n, &mb, &nb, &i_zero, &i_zero, &contxt,
            &local_rows_a, &info);
  create_block_cyclic_matrix_blacs(m, n, mb, nb, a, lda,
                  local_a, contxt);
  pdgetrf_(&m, &n, local_a, &i_one, &i_one, DESCA, ipiv, &info);
  Cblacs_exit(0);
}
```

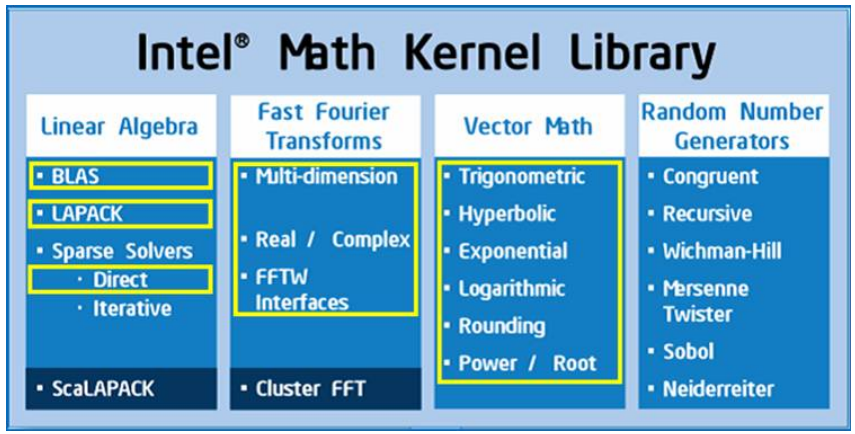
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



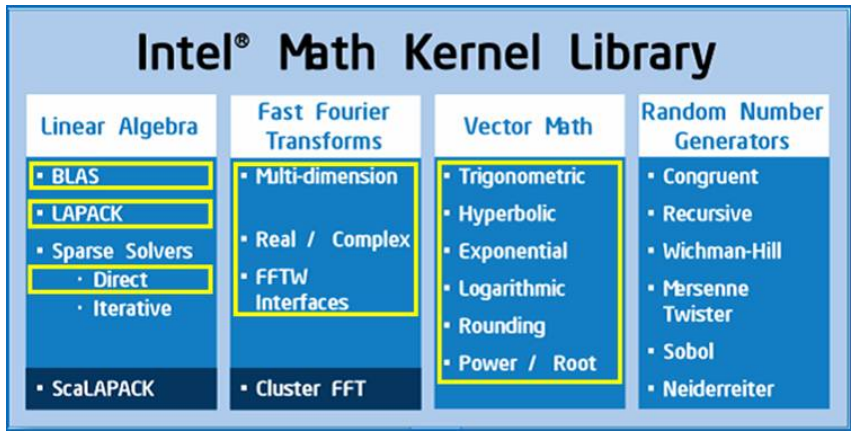
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



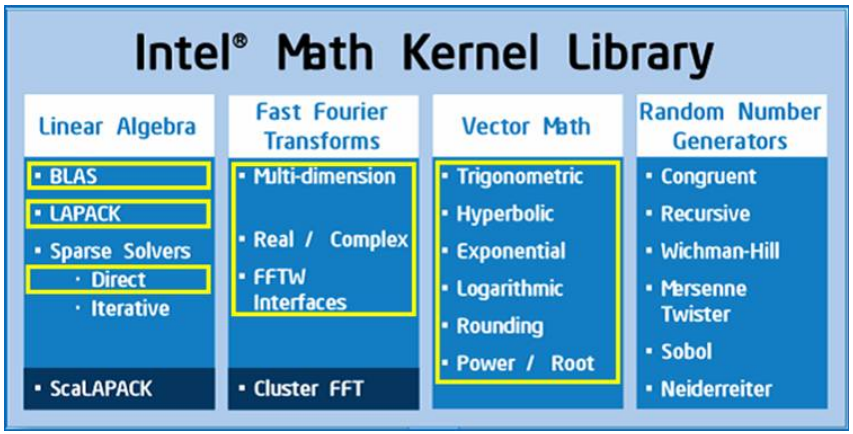
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



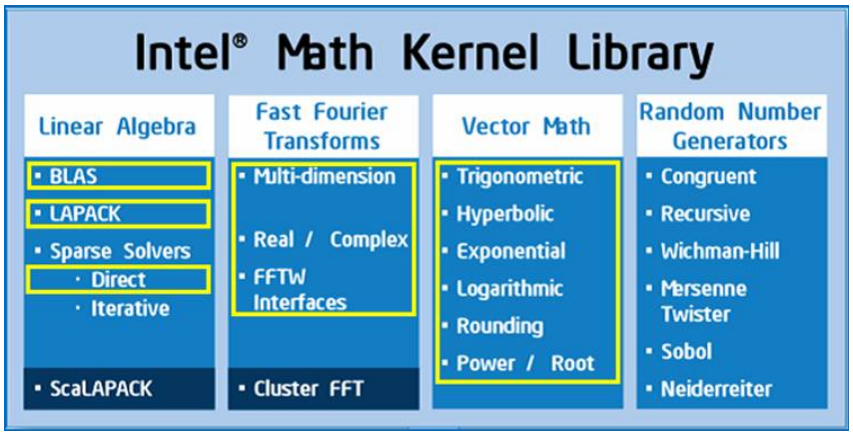
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



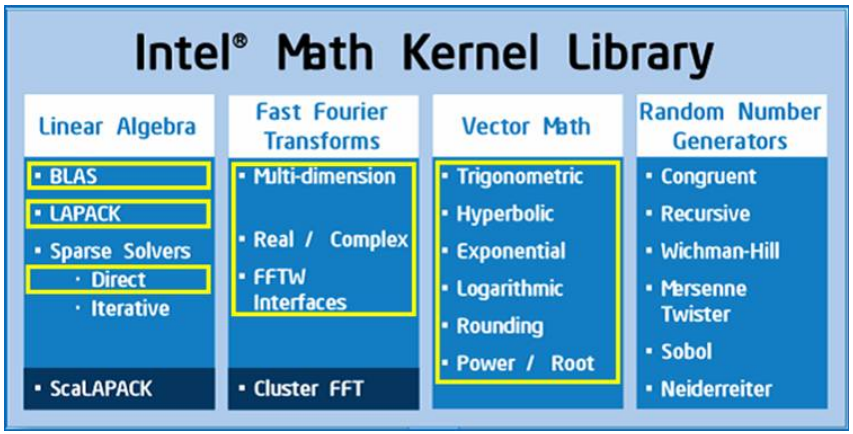
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



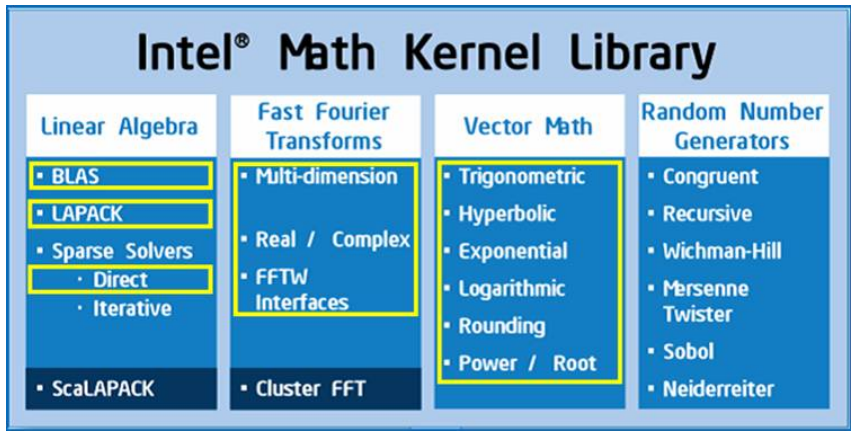
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



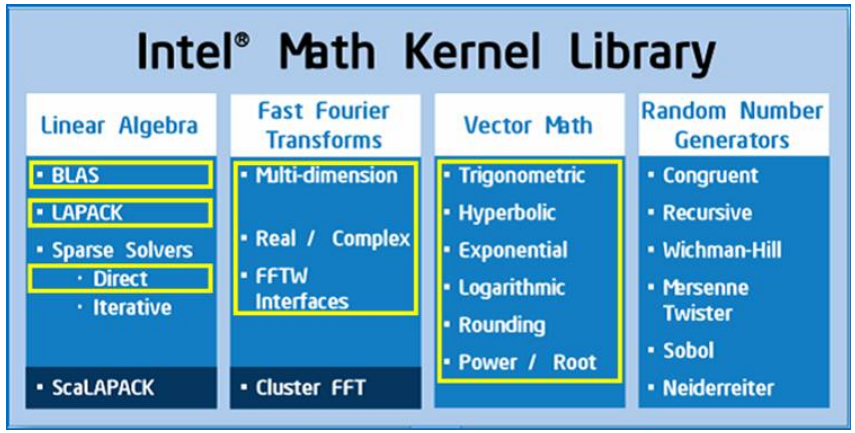
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



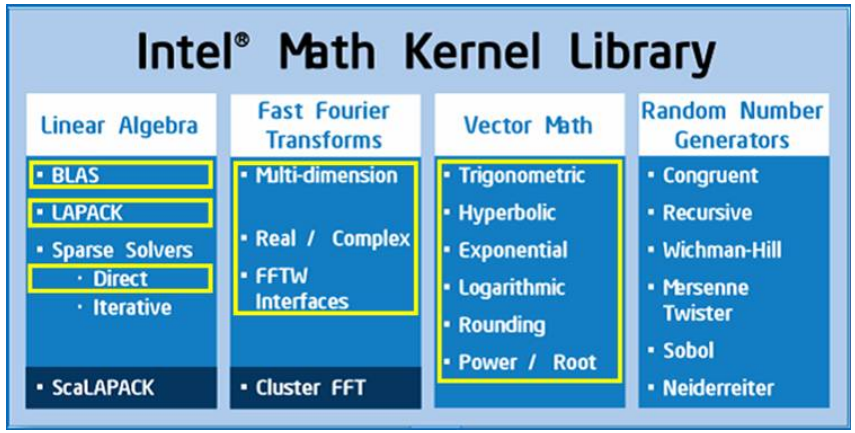
# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código
  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.



# Explotación paralelismo

- Hay bastantes oportunidades para explotar el paralelismo:
  - Nivel 3 BLAS $O(n^3)$
  - LAPACK $O(n^3)$
  - FFT $O(n \log n)$
- Implementación multi-hilo utilizando OpenMP
- Se soporta los compiladores de Intel y GCC
- No es necesario reescribir el código




  - ScaLAPACK también se puede utilizar en entornos SMP (memoria compartida)
  - Todas las rutinas MKL son thread-safe. Prepararadas para utilizarse desde OpenMP.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Lincado con Intel MKL

Modelo por capas de la librería.

- Capa de Interface
  - Compilador: Intel / GNU
  - LP64 / ILP64
- Capa de Threading
  - Compatibilidad con la implementación de hilos del compilador.
  - Secuencial
- Capa Computacional
  - Selección del código según la arquitectura (IA-32, Intel 64, IA-64)
- Capa Run-Time
  - Compatibilidad OpenMP con el compilador.

# Ejemplos lincado con Intel MKL

Ejemplo 1: Lincado estático utilizando Intel FORTRAN Compiler, BLAS, Intel 64, Linux:

```
ifort myprog.f libmkl_intel_lp64.a libmkl_intel_thread.a \
      libmkl_core.a  libiomp5.so
```

Ejemplo 2: Lincado dinámico con Intel C/C++ Compiler, BLAS, Intel 64, Linux

```
icc myprog.f -lmkl_intel_ilp64 -lmkl_intel_thread \
    -lmkl_core -liomp5 -lpthread
```

Ejemplo 3: Lincado dinámico con Intel C/C++ Compiler, ScaLAPACK, Intel 64, Linux

```
mpiicc myprog.c -lmkl_scalapack_ilp64 -lmkl_blacs_intelmpi_ilp64 \
      -lmkl_lapack -lmkl_intel_ilp64 \
      -lmkl_intel_thread -lmkl_lapack \
      -lmkl_core -liomp5 -lpthread
```

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Entorno de trabajo en Ben-Arabi

- Entorno modules para establecer las variables de entorno que permiten acceder a los compiladores, librerías y aplicaciones instaladas en Ben-Arabi.
  - `module av`
  - `module list`
  - `module load icc`
  - `module load ifort`
  - `module load mkl`
  - `module load impi`
- Sistema de gestión de trabajos de usuario LSF.
  - `bqueues`
  - `bsub`
  - `bjobs`
  - `bkill`
  - `bpeek`

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios I

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# *Ejemplos y Ejercicios I*

En las prácticas con el SuperComputador Ben-Arabi:

- BLAS: Multiplicación de matrices y comparación de tiempos de ejecución
  - Con un triple bucle
  - Sustituir el bucle más interno con una llamada a la rutina `DDOT` de BLAS nivel 1.
  - Sustituir los dos bucles más internos con una llamada a la rutina `DGEMV` de BLAS nivel 2.
  - Con la rutina de multiplicación de matrices `DGEMM` de BLAS nivel 3.
  - Versión paralela de `DGEMM`. Ejecución con 4 y 8 hilos.
- LAPACK: Factorización y Resolución de Sistema de Ecuaciones.
  - Utilizando rutinas computacionales (`DGETRF`, `DGETRS`) y rutinas driver (`DGESV`).
  - Versión paralela. Ejecución con 4 y 8 hilos. Comparación tiempos de ejecución.

# Ejemplos y Ejercicios II

- PBLAS y ScaLAPACK
  - Rutina de multiplicación de matrices `PDGEMM` de PBLAS. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con BLAS nivel 3.
  - Utilizando rutinas computacionales `PDGETRS`. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma de la malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con LAPACK.

# Ejemplos y Ejercicios II

- PBLAS y ScaLAPACK
  - Rutina de multiplicación de matrices `PDGEMM` de PBLAS. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con BLAS nivel 3.
  - Utilizando rutinas computacionales `PDGETRS`. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma de la malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con LAPACK.

# Ejemplos y Ejercicios II

- PBLAS y ScaLAPACK
  - Rutina de multiplicación de matrices `PDGEMM` de PBLAS. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con BLAS nivel 3.
  - Utilizando rutinas computacionales `PDGETRS`. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma de la malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con LAPACK.

# *Ejemplos y Ejercicios II*

- PBLAS y ScaLAPACK
  - Rutina de multiplicación de matrices `PDGEMM` de PBLAS. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con BLAS nivel 3.
  - Utilizando rutinas computacionales `PDGETRS`. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma de la malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con LAPACK.

# *Ejemplos y Ejercicios II*

- PBLAS y ScaLAPACK
  - Rutina de multiplicación de matrices `PDGEMM` de PBLAS. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con BLAS nivel 3.
  - Utilizando rutinas computacionales `PDGETRS`. Comparación tiempos de ejecución con diferentes tamaños de bloque, forma de la malla 2D de procesos con 8 y 16 procesos.
  - Comparación tiempos de ejecución con LAPACK.