TFC – Arquitectura de Computadors i Sistemes Operatius

Títol: Paral·lelització de la factorització LU utilitzant OpenCL

Alumne: Javier Vegas Caballero (jvegasca@uoc.edu)

Consultor: Francesc Guim Bernat

UOC

Universitat Oberta de Catalunya

Índex

UOC

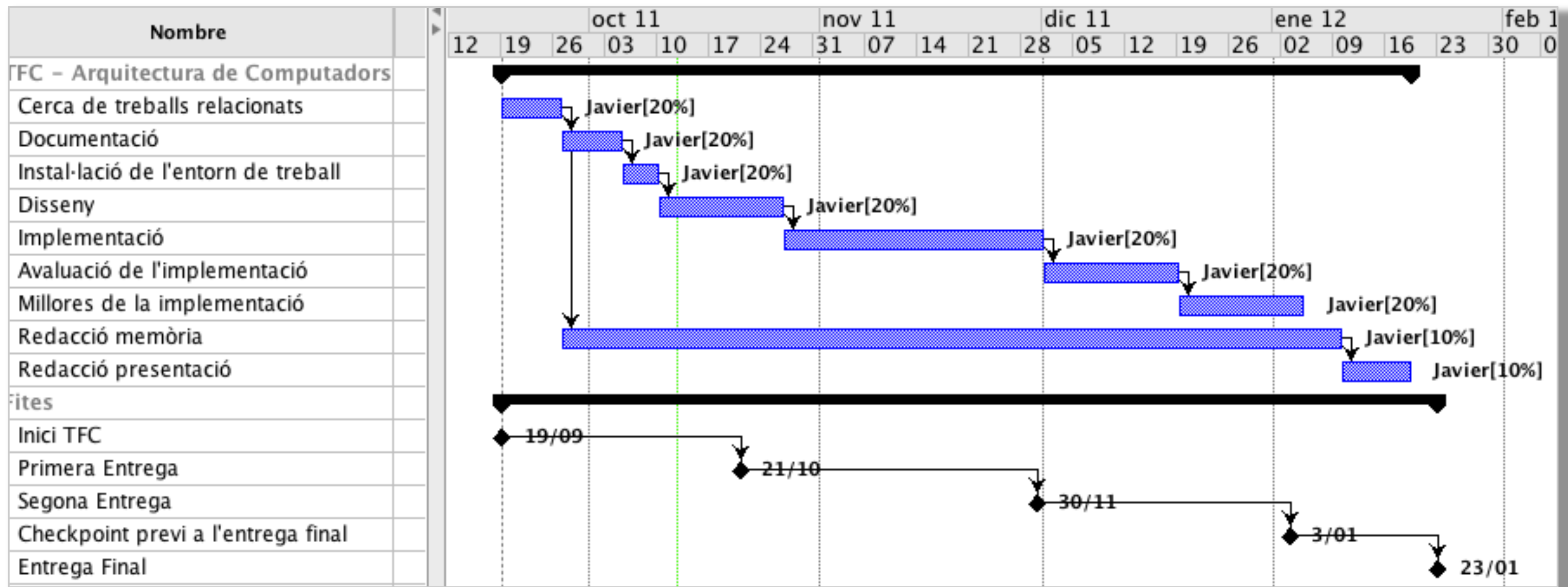
- Objectius i Motivació
- Planificació
- OpenCL
- Factorització LU
- Implementacions
- Resultats
- Possibles millores
- Conclusions

Objectius i Motivació

UOC

- Els objectius del projecte són els següents:
  - L'estudi de l'estàndard OpenCL (Open Computing Language)
  - La seva avaluació desenvolupant la paral·lelització d'una factorització LU.
- La motivació principal és conèixer la programació paral·lela a l'entorn de la computació amb dispositius GPU.

Planificació



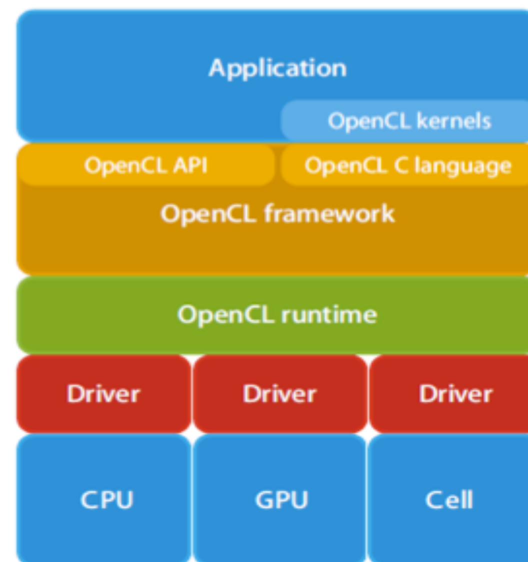
OpenCL (1)

UOC

- L'*Open Computing Language* (OpenCL) és un estàndard obert per a la programació d'entorns heterogenis on coexisteixen diferents recursos de computació: CPUs i GPUs.
- Una aplicació OpenCL es la que utilitza durant la seva execució les funcionalitats especifiques de l'OpenCL per portar a terme les tasques computa-cionals en les unitats de computació disponibles al sistema on s'executa.

OpenCL (2)

- Arquitectura

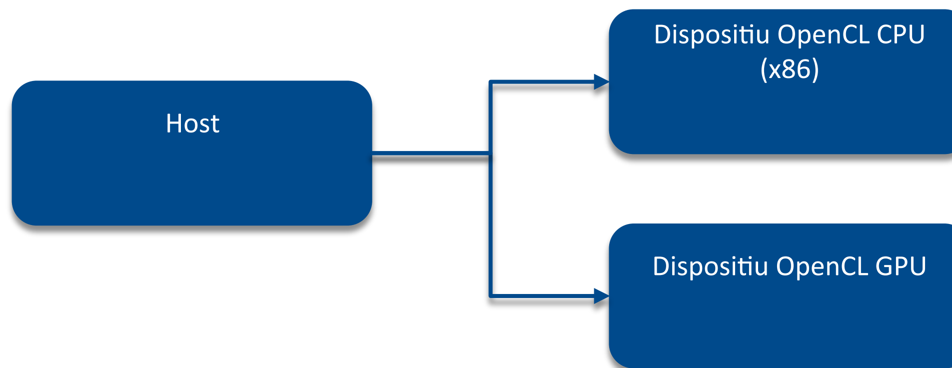


- Elements que intervenen en l'execució d'aplicatius OpenCL distribuïts en capes.

OpenCL (3)

UOC

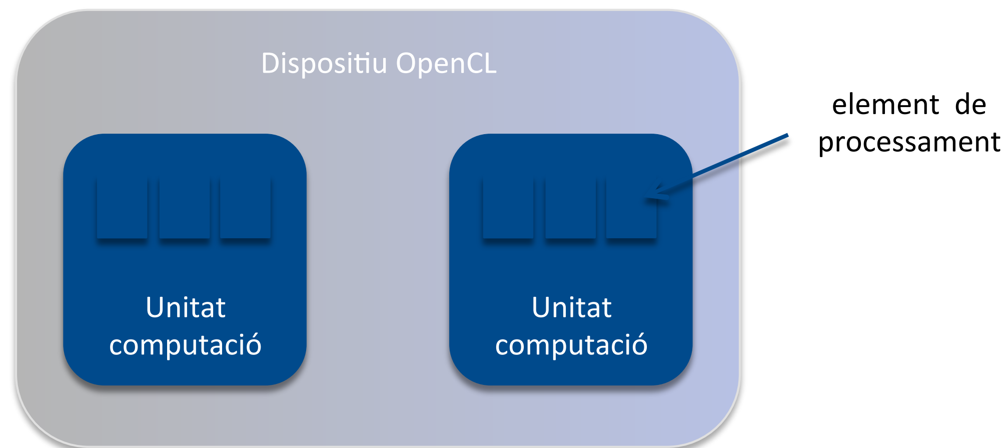
- Interacció Host – Dispositiu



- El host és l'encarregat d'iniciar l'execució de l'aplicació OpenCL.
- La comunicació entre el host i els dispositius es realitza mitjançant el enviament de comandes.

OpenCL (4)

- Estructura d'un dispositiu OpenCL



- Els dispositius estan formats per unitats de computació que alhora es divideixen en elements de processament. Aquests darrers són els encarregats de portar a terme l'execució dels kernels.

Factorització LU

- És un mètode per descompondre una matriu A (no singular) en dos matrius una triangular inferior L i una altre triangular superior U .

A = L· U

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Implementacions (1)

UOC

- Versió seqüencial

```
:

/* Per a cada iteració i */
for(i=0;i<n;i++)
{
            /* Assignem a Lii=1 */
      L[i*n+i]=1.0f;

      /* Per a cada fila j de la matriu A */
            for(j=i+1;j<n;j++)
            {
                        L[j*n+i]=U[j*n+i]/U[i*n+i];

                        /* Per a cada columna k de la matriu A */
            for(k=i;k<n;k++)
                  U[j*n+k]-=(L[j*n+i]*U[i*n+k]);
      }
}

:
```

- Cal que la matriu U s'inicialitzi amb els elements de la matriu A.
- Es fa servir una indexació de matrius com a vectors.

Implementacions (2)

- Versió paral·lela (inicial)
  - Es crida dins un bucle principal dos kernels (GPU):
    - El kernel 1 s'encarrega d'omplir la matriu L.

```
kernel 1

Fitxer: Pivot.cl

__kernel void Pivot(__global float* AU,__global float* L, const int n, const int i)
{

    int j=get_global_id(0)+i;

    if(j>n-1)
        return;

    L[j*n+i]=native_divide(AU[j*n+i],AU[i*n+i]);


}
```

Implementacions (3)

- Versió paral·lela (inicial)
  - El kernel 2 s'encarrega d'omplir la matriu U.

```
kernel 2

Fitxer: ForwardElimination.cl

__kernel void ForwardElimination(__global float* AU, __global float* L, const int
n, const int i)
{

    int j=get_global_id(0)+(i+1);
    int k=get_global_id(1)+i;

    if(j>n-1 || k>n-1)
        return;

    AU[j*n+k]-=(L[j*n+i]*AU[i*n+k]);

}
```

  - El vector AU representa la matriu U.

Implementacions (4)

UOC

- Versió paral·lela (inicial)
  - Exemple on A es una matriu 5x5

0				
0	0			
0	0	0		
0	0	0	0	

matriu U

0	0	0	0	0
	0	0	0	0
		0	0	0
			0	0
				0

matriu L

Implementacions (5)

- Versió paral·lela (millorada)
  - La versió millorada modifica el següent:
    - En lloc d'executar dos kernels per iteració es passa a treballar amb nomes un.
    - El nou kernel només treballa amb un espai d'indexat unidimensional, això vol dir que cada work-item processarà una fila sencera de la matriu.
    - Fa ús de memòria local.

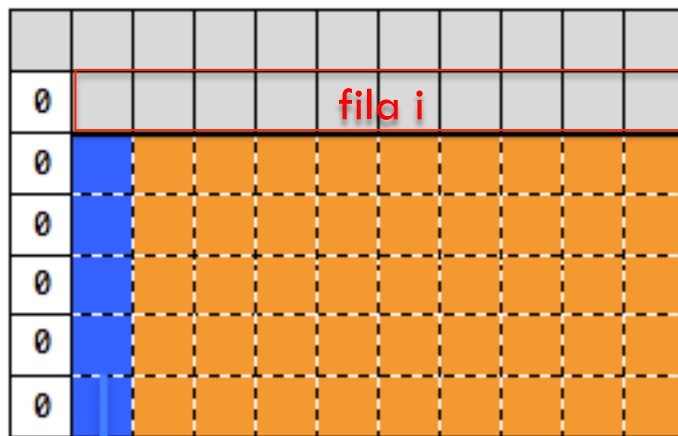
Implementacions (6)

- Versió paral·lela (millorada)
  - El nou kernel és el següent:

```
__kernel void ludecomposition(__global float* A, const int n, const int i,
__local float* Ai)
{

    int k;
    float ratio;

    int j=get_global_id(0);
    int jloc=get_local_id(0);
    int nloc=get_local_size(0);

    if(j>n-1)
        return;

    ratio=native_divide(A[j*n+i],A[i*n+i]);

    for(k=i+jloc;k<n;k=k+nloc)
        Ai[k]=A[i*n+k];

    barrier(CLK_LOCAL_MEM_FENCE);

    if(j<=i)
        return;

    for(k=i;k<n;k++)
        A[j*n+k]-=(ratio*Ai[k]);

    A[j*n+i]=ratio;

}
```

Implementacions (7)

- Versió millorada
  - Per a una iteració k:



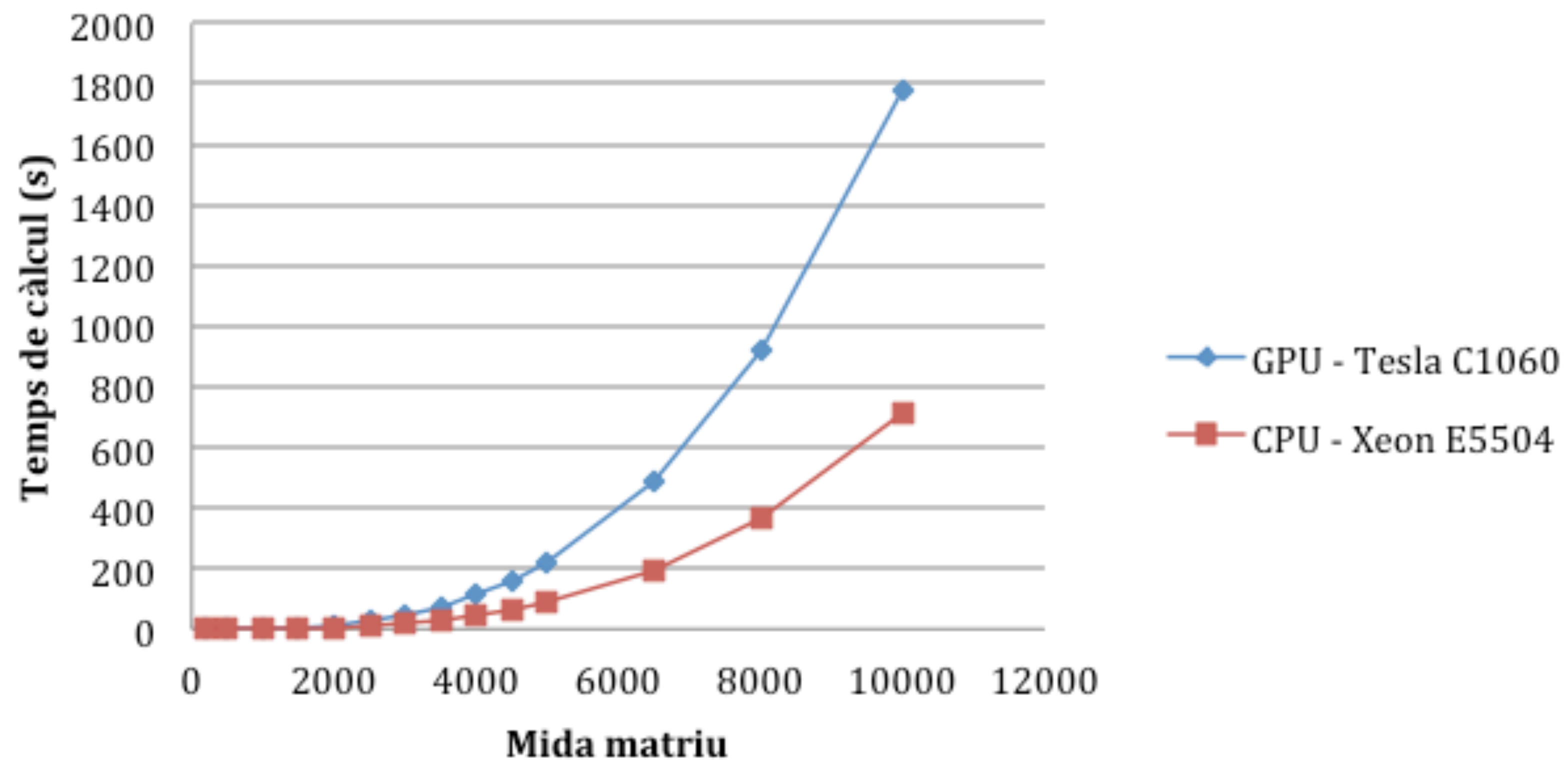
work-group

submatriu processada a la iteració k

  - Un work-item s'encarrega de processar una fila sencera.
  - Els work-items treballen en paral·lel per copiar els elements d'una *fila i* al vector Ai (memòria local).

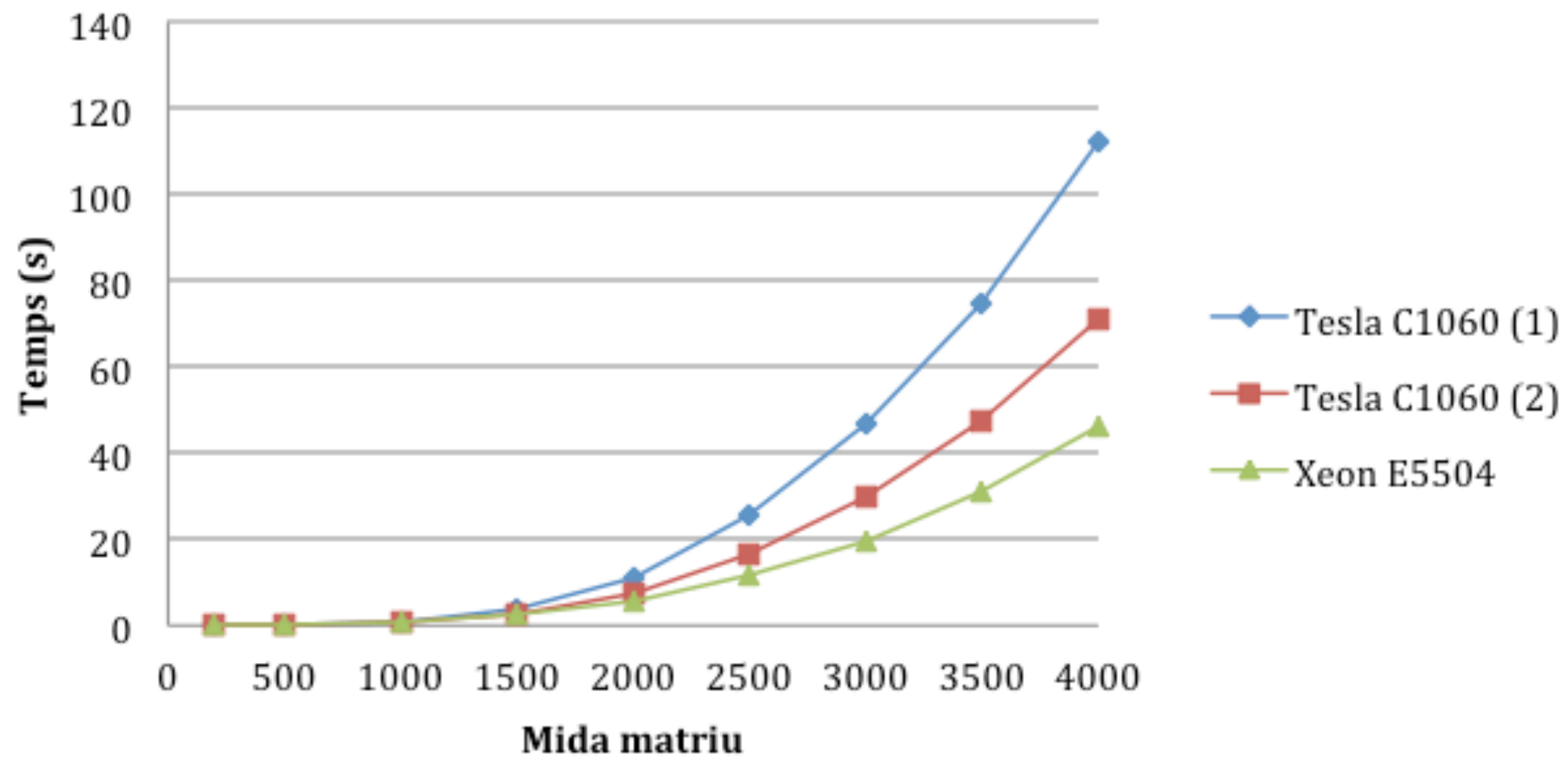
Resultats (1)

UOC



Resultats (2)

UOC



UOC

Possibles millores (1)

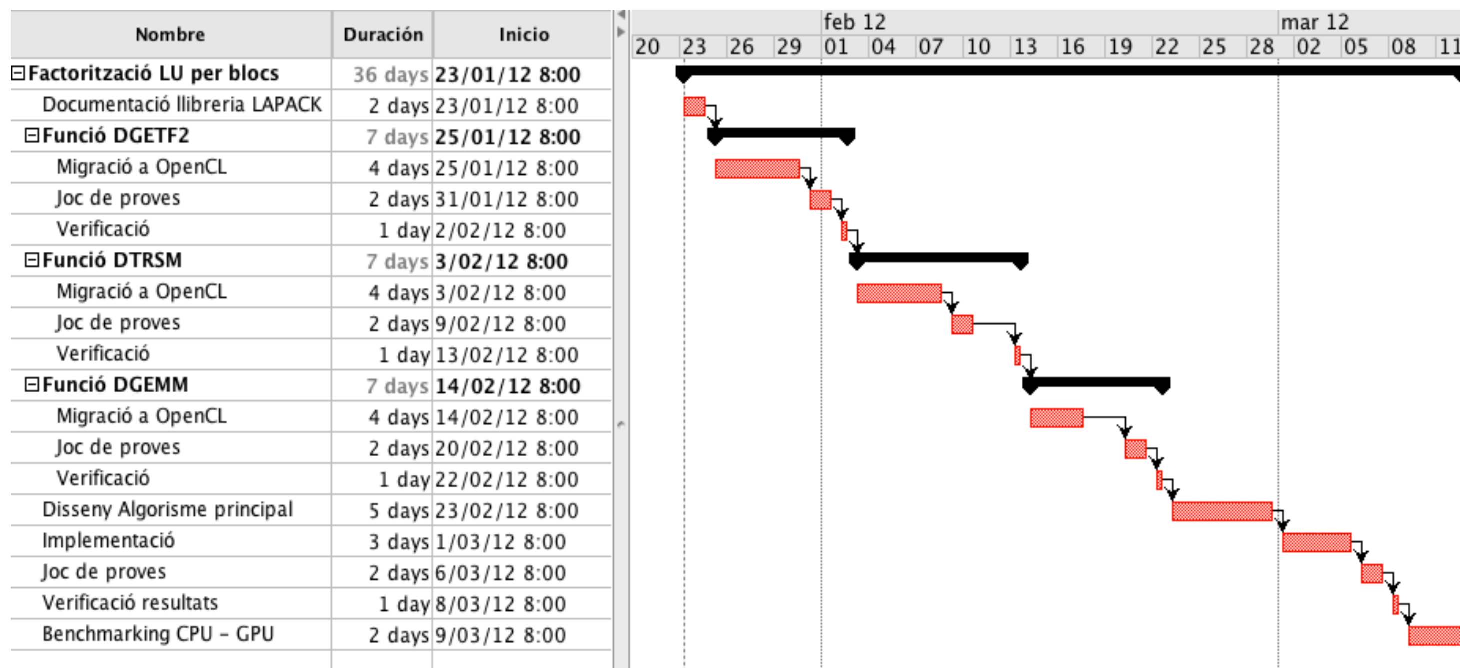
- Factorització LU per blocs

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} = \begin{bmatrix} L_{00} & 0 \\ L_{10} & L_{11} \end{bmatrix} \times \begin{bmatrix} U_{00} & U_{01} \\ 0 & U_{11} \end{bmatrix}$$

  - La implementació d'aquest procediment a OpenCL és força complexa donat que per a resoldre eficientment caldria disposar d'una versió de la llibreria LAPACK implementada sobre OpenCL.

Possibles millores (2)

- Possible planificació amb les tasques a realitzar per implementar a OpenCL la factorització per blocs



Nombre	Duración	Inicio
⊟Factorització LU per blocs	36 days	**23/01/12 8:00**
Documentació llibreria LAPACK	2 days	23/01/12 8:00
⊟Funció DGETF2	7 days	**25/01/12 8:00**
Migració a OpenCL	4 days	25/01/12 8:00
Joc de proves	2 days	31/01/12 8:00
Verificació	1 day	2/02/12 8:00
⊟Funció DTRSM	7 days	**3/02/12 8:00**
Migració a OpenCL	4 days	3/02/12 8:00
Joc de proves	2 days	9/02/12 8:00
Verificació	1 day	13/02/12 8:00
⊟Funció DGEMM	7 days	**14/02/12 8:00**
Migració a OpenCL	4 days	14/02/12 8:00
Joc de proves	2 days	20/02/12 8:00
Verificació	1 day	22/02/12 8:00
Disseny Algorisme principal	5 days	23/02/12 8:00
Implementació	3 days	1/03/12 8:00
Joc de proves	2 days	6/03/12 8:00
Verificació resultats	1 day	8/03/12 8:00
Benchmarking CPU – GPU	2 days	9/03/12 8:00

Conclusions

UOC

- No tots els algorismes són directament paral·lelitzables, el cas tractat de la factorització és un exemple ja que n'hi han diverses formes de resoldre-ho però no totes son adients per ser implementades sobre el llenguatge OpenCL. La resolució de una factorització LU és un exemple clar.
- He trobat serioses deficiències a l'entorn de desenvolupament utilitzat (plataforma Mac OS X) més concretament respecte a l'entorn de depuració (*debugging*). La possibilitat de depurar fent l'execució pas a pas dins d'un kernel han sigut nul·la. Tot això ha fet que la part de depuració sigui més costosa de lo inicialment planificat.
- Com a resultat de l'estudi del llenguatge OpenCL i de les proves realitzades durant l'aprenentatge he comprovat l'augment de rendiment que es pot proporcionar la utilització de la GPU com a recurs de computació.
- Les implementacions d'OpenCL existents no son completament homogènies.