LASs – Linear Algebra routines on OmpSs

Sandra Catalán (scatalan@bsc.es), Pedro Valero, Xavier Martorell and Jesús Labarta
1) Introduction
2) TRSM in LASs
3) NPGETRF in LASs
4) sLASs on small mat.
5) SpMV
6) Ongoing work
**Programming Model**

**OmpSs and OpenMP Tasks (Runtime):**

- We divide the problem into tasks
  - Data dependences among tasks (DAG)
- Runtime manages and schedules the running of the tasks
  - Transparent
    - Runtime is in charge of the optimizations, not the programmer
  - Easy to implement
  - Easy to port
  - Easy to maintain

```c
ddss_dgemm( ... )
{
  ...
  for ( mi = 0; mi < mt; mi++ ){
    for ( ni = 0; ni < nt; ni++ ){
      for ( ki = 0; ki < kt; ki++ ){
        #pragma oss task in( TILE_A[mi][ki] ) \ 
          in( TILE_B[ki][ni] ) \ 
          inout( TILE_C[mi][ni] ) \ 
          shared( TILE_A, TILE_B, TILE_C ) \ 
          firstprivate( mi, ni, ki, betat )
        cblas_dgemm( CblasRowMajor,
                    ( CBLAS_TRANSPOSE ) TRANS_A,
                    ( CBLAS_TRANSPOSE ) TRANS_B,
                    tile_size_m,
                    tile_size_n,
                    tile_size_k,
                    ALPHA, TILE_A[mi][ki], tile_size_k,
                    TILE_B[ki][ni], tile_size_n,
                    betat, TILE_C[mi][ni], tile_size_n );
      }
    }
  }
}
```
OpenMP vs OmpSs

```
#pragma omp parallel
#pragma omp master
{
  #pragma omp task depend(in:A[0:lda*ak]) \
    depend(in:B[0:ldb*bk]) \
    depend(out:C[0:ldc*n])
  {
    core_dgemm(transa, transb, \
               m, n, k, \
               alpha, A, lda, \
               B, ldb, \
               beta, C, ldc);
  }
  #pragma omm taskwait
}
```

Diagram:
- OpenMP PARALLEL (FORK)
- END OF PARALLEL REGION
- IMPLICIT SYNCHRONIZATION
- Task pool
- Runtime
- Thread 1
- Thread 2
- Thread n
1) Introduction

2) TRSM in LASs

3) NPGETRF in LASs

4) sLASs on small mat.

5) SpMV

6) Ongoing work
BLAS-3 Optimization

dtrsm (triangular solve)

• Join in one tasks multiple \textit{dgemm} that compute one column, instead of one task per tile
  ∗ Less number of tasks/dependences
  ∗ Smaller DAG
  ∗ Better exploitation of memory hierarchy
  ∗ Tunable (Auto-tuning)
  ∗ \textbf{Use of OmpSs regions}
  ∗ Tile Size \rightarrow size of the matrix and #cores
  * \(36864^2 \rightarrow 768^2\)

```c
for (d = 0; d < dt; d++){
  for (c = 0; c < ct; c++){
    #pragma oss task ...
    dtrsm( ... );
  }
  for (c = 0; c < ct; c++) {
    for (r = d; r < rt; r++)  {
      #pragma oss task ...
      dgemm( ... );
    }
  }
}
```
Performance Evaluation (Extrae + Paraver)

dtrsm
- LASs-opt
  - Faster than LASs
    - from $18432^2$ in dtrsm and from $24576^2$ in dtrmm
    - $5\% (18432^2) \rightarrow 12\% (36864^2)$
  - Faster than Plasma (OpenMP)
    - from $30720^2$
    - $12\% (30720^2) \rightarrow 15\% (36864^2)$
  - On small matrices the optimization is not effective
    - $6144^2 \rightarrow \text{tile\_size} = 128$
    - Small tiles $\rightarrow$ small IPC (GLFOPS) in dgemm

![Graph showing performance comparison](image)

<table>
<thead>
<tr>
<th>Matrix Size (MxM)</th>
<th>Time(s)</th>
<th>Plasma</th>
<th>LASs</th>
<th>LASs_opt</th>
</tr>
</thead>
<tbody>
<tr>
<td>6144</td>
<td></td>
<td>0.16</td>
<td>0.20</td>
<td>0.29</td>
</tr>
<tr>
<td>12800</td>
<td></td>
<td>1.12</td>
<td>1.45</td>
<td>1.50</td>
</tr>
<tr>
<td>18432</td>
<td></td>
<td>3.96</td>
<td>4.28</td>
<td>4.06</td>
</tr>
<tr>
<td>24576</td>
<td></td>
<td>9.20</td>
<td>9.83</td>
<td>9.44</td>
</tr>
<tr>
<td>30720</td>
<td></td>
<td>19.89</td>
<td>19.83</td>
<td>17.45</td>
</tr>
<tr>
<td>36864</td>
<td></td>
<td>38.29</td>
<td>36.61</td>
<td>32.29</td>
</tr>
</tbody>
</table>
sLASs

dtrsm (triangular solve)

- Depending on the matrix size and number of cores, the optimization is effective or not
  → We need a mechanism that allows us to adapt the execution, depending on:
    ▪ the size of the matrix
    ▪ the cores available
  → Minimum modifications on the code
    ▪ Low maintainability cost
    ▪ Portability and adaptability

- This is implemented by using:
  → Regions
  → Weak dependences
  → Final clause

```c
for ( d = 0; d < dt; d++){
    for ( c = 0; c < ct; c++){
        #pragma oss task ...
        dtrsm( … );
    }
    for ( c = 0; c < ct; c++) {
        for ( r = d; r < rt; r++)  {
            #pragma oss task …
            dgemm( … );
        }
    }
}
```

```c
smart_dtrsm(is_final);
for ( d = 0; d < dt; d++){
    for ( c = 0; c < ct; c++){
        #pragma oss task ...
        dtrsm( … );
    }
    for ( c = 0; c < ct; c++) {
        #pragma oss task \ 
        weakinout (TILEB[d:rt-1][c]) …
        final(is_final)
        for ( r = d; r < rt; r++) {
            #pragma oss task \ 
            inout(TILEB[r][c]) …
            dgemm( … );
        }
    }
}
```

Barcelona Supercomputing Center  
Centro Nacional de Supercomputación
Performance Evaluation (Extrae + Paraver) dtrsm@sLASs

- Obtains similar results to the reference implementations
  - Nesting (condition of the final OmpSs clause = false):
    - from $6144^2$ to $18432^2$
      - Insufficient parallelism to do join on dgemm
      - $6144 / 512$ (Default tile size) $<$ number of cores (48)
    - Instantiate more tasks
      - $0.82\% (18432^2)$
  - No nesting (condition of the final OmpSs clause = true)
    - From $24576^2$ to $36864^2$
      - Big matrices $\rightarrow$ join on dgemm and bigger tiles size
    - No overhead

<table>
<thead>
<tr>
<th>Matrix Size (MxM)</th>
<th>LASs</th>
<th>LASs_opt</th>
<th>sLASs</th>
</tr>
</thead>
<tbody>
<tr>
<td>6144</td>
<td>0.20</td>
<td>0.29</td>
<td>0.20</td>
</tr>
<tr>
<td>12288</td>
<td>1.45</td>
<td>1.54</td>
<td>1.54</td>
</tr>
<tr>
<td>18432</td>
<td>4.28</td>
<td>4.06</td>
<td>4.35</td>
</tr>
<tr>
<td>24576</td>
<td>9.83</td>
<td>9.44</td>
<td>9.51</td>
</tr>
<tr>
<td>30720</td>
<td>19.83</td>
<td>17.45</td>
<td>17.36</td>
</tr>
<tr>
<td>36864</td>
<td>36.61</td>
<td>32.29</td>
<td>30.63</td>
</tr>
</tbody>
</table>

Time(s)
1) Introduction
2) TRSM in LASs
3) NPGETRF in LASs
4) sLASs on small mat.
5) SpMV
6) Ongoing work
sLASs

dnpgetrf (LU factorization without pivoting)

- Np → no pivoting
  - Well-conditioned matrices
  - For the sake of performance evaluation
    - To compare with pivoting versions
- Dynamic behavior
  - The parallelism (# dgemm) is reduced along the factorization
    - In the first steps → the optimization can be effectively exploited
    - In the last steps → the optimization is not effective
  - The “is_final” condition must be computed every step
    - smart_dnpgetrf

```c
for ( d = 0; d < dt; d++){
    #pragma oss task ...
    dnpgetrf( ... );
    for ( r = d+1; r < rt; r++){
        #pragma oss task ...
        dtrsm( ... );
    }
    for ( c = d+1; c < ct; c++){
        #pragma oss task ...
        dtrsm( ... );
    }
    for ( c = d+1; c < ct; c++){
        smart_dnpgetrf(is_final);
        #pragma oss task \ 
        weaken(TILEA[d+1:rt-1][d]) ... 
        final(is_final) 
        for ( r = d+1; r < rt; r++) {
            #pragma oss task \ 
            in(TILEA[r][d]) ... 
            dgemm( ... );
        }
    }
}
```
Performance Evaluation (Extrae + Paraver)
dnpgetrf@sLASs

- Nesting (condition of the final OmpSs clause = false):
  - from 6144\(^2\) to 18432\(^2\)
    - Insufficient parallelism to do join on dgemm
    - 6144 / 512 (Default tile size) < number of cores (48)
  - Instantiate more tasks
    - 0.79\% (12288\(^3\))

- No nesting (condition of the final OmpSs clause = true)
  - From 24576\(^2\) to 36864\(^2\)
    - In the first steps → high parallelism
    - Big matrices → join on dgemm and bigger tiles size
  - Time Reduction in the tasks instantiation
    - 36.03\% → 17.31\%
  - Increase of IPC in dgemm
    - ~1.7 → ~2.25

<table>
<thead>
<tr>
<th>Matrix Size (MmM)</th>
<th>6144</th>
<th>12288</th>
<th>18432</th>
<th>24576</th>
<th>30720</th>
<th>36864</th>
</tr>
</thead>
<tbody>
<tr>
<td>LASs</td>
<td>0.19</td>
<td>0.93</td>
<td>2.90</td>
<td>6.63</td>
<td>13.10</td>
<td>22.40</td>
</tr>
<tr>
<td>sLASs</td>
<td>0.19</td>
<td>0.93</td>
<td>2.96</td>
<td>7.04</td>
<td>11.45</td>
<td>20.28</td>
</tr>
</tbody>
</table>
1) Introduction
2) TRSM in LASs
3) NPGGETRF in LASs
4) sLASs on small mat.
5) SpMV
6) Ongoing work
SLASs – Improving small matrices performance

Why small matrices?

- SLASs
  - $N=12288$, $b=512$, 48 cores
- There exists a significant workload imbalance (20% time)
- Improving workload balance benefits from small to large matrices.
SLASs – Improving small matrices performance

From LASs to sLASs

- **SLASs**
  - $N=4096$, $b=512$, 48 cores
  - Big workload imbalance.

- **SLASs + opt. gemm/trsm tasks**
  - $N=4096$, $b=512$, 48 cores
  - Workload imbalance is considerably reduced.

- **PLASMA**
  - $N=4096$, $b=512$, 48 cores
  - Big workload imbalance.
1) Introduction
2) TRSM in LASs
3) NPGETRF in LASs
4) sLASs on small mat.
5) SpMV
6) Ongoing work
LASs
dspmv (sparse matrix-vector multiplication) Y = ALPHA * A * X + Y * BETA

1 Task per Row
- High number of tasks
- Tasks are not big enough

Blocking
- Requires preprocessing
- Data dependences

Taskloop
- Grainsize needs to be determined
- Easy to implement

Blocking + Nesting
- Requires preprocessing
- Data dependences

Grouping
- Requires extra calculations to create groups
#pragma oss taskloop grainsize ...
for ( r = 0; r < nRows; r++){
   Sval = 0.0;
   for ( c = 0; c < nCols; c++) {
      val = VAL_A[ROW_A[r]+c];
      col = COL_A[ROW_A[r]+c];
      sval += val* X[col]*ALPHA;
   }
   Y[r] = sval + Y[r] *BETA;
}

for ( rb = 0; rb < nRows; rb++)
{
  #pragma oss task
  {
   nnz = number_of_non_zeros_in_row_block(rb);
   #pragma omp taskloop num_tasks(nnz/th) if (nnz > th)
   for ( r = rb; r <r + rb ; r++){
      Sval = 0.0;
      for ( c = 0; c < nCols; c++) {
         val = VAL_A[ROW_A[r]+c];
         col = COL_A[ROW_A[r]+c];
         sval += val* X[col]*ALPHA;
      }
      Y[r] = sval + Y[r] *BETA;
   }
  }
}

Note that th is a threshold set to 40,000, a number determined experimentally.
Ongoing work

• Iterative sparse solvers
  • CG

• Direct sparse solvers

• Improving GESV performance
  • Small matrices optimizations