Comparative Study of One-Sided Factorizations with Multiple Software Packages on Multi-Core Hardware

Emmanuel AGULLO
Jack DONGARRA
Bilel HADRI
Jakub KURZAK
Hatem LTAIEF
Piotr LUSZCZEK

Scheduling for Large-Scale Systems, Knoxville, TN, May 13-15, 2009

Outline

1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

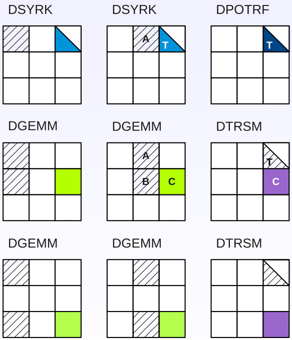
Outline

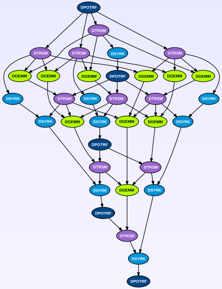
1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

Outline

Tile Cholesky Factorization

```
FOR k = 0..TILES-1
  FOR n = 0..k-1
    A[k][k] ← DSYRK(A[k][n], A[k][k])
  A[k][k] ← DPOTRF(A[k][k])
  FOR m = k+1..TILES-1
    FOR n = 0..k-1
      A[m][k] ← DGEMM(A[k][n], A[m][n], A[m][k])
    A[m][k] ← DTRSM(A[k][k], A[m][k])
```



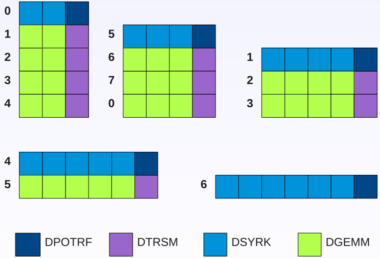


- Basically identical to the block algorithm (LAPACK).
- Input matrix stored and processed by square tiles.
- Complex DAG.

Tile Cholesky Factorization - Static pipeline

```
void dsyrk(double *A, double *T);
void dpotrf(double *T);
void dgemm(double *A, double *B, double *C);
void dtrsm(double *T, double *C);


k = 0; m = my_core_id;
while (m >= TILES) {
   k++; m = m-TILES+k;
} n = 0;

while (k < TILES && m < TILES) {
   next_n = n; next_m = m; next_k = k;

   next_n++;
   if (next_n > next_k) {
      next_m += cores_num;
      while (next_m >= TILES && next_k < TILES) {
         next_k++; next_m = next_m-TILES+next_k;
      } next_n = 0;
   }

   if (m == k) {
      if (n == k) {
         dpotrf(A[k][k]);
         core_progress[k][k] = 1;
      }
      else {
         while(core_progress[k][n] != 1);
         dsyrk(A[k][n], A[k][k]);
      }
   }
   else {
      if (n == k) {
         while(core_progress[k][k] != 1);
         dtrsm(A[k][k], A[m][k]);
         core_progress[m][k] = 1;
      }
      else {
         while(core_progress[k][n] != 1);
         while(core_progress[m][n] != 1);
         dgemm(A[k][n], A[m][n], A[m][k]);
      }
   }
```

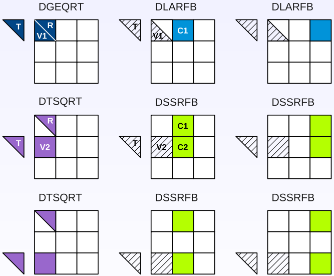
- Work partitioned in one dimension (by block-rows).
- Cyclic assignment of work across all steps of the factorization (pipelining of factorization steps).
- Process tracking by a global progress table.
- Stall on dependencies (busy waiting).

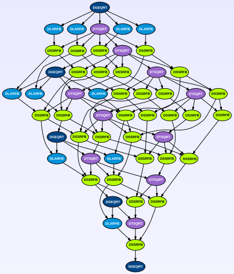


Outline

Tile QR (&LU) Factorization

```
FOR k = 0..TILES-1
  A[k][k], T[k][k] ← DGRQRT(A[k][k])
  FOR m = k+1..TILES-1
    A[k][k], A[m][k], T[m][k] ← DTSQRT(A[k][k], A[m][k], T[m][k])
  FOR n = k+1..TILES-1
    A[k][n] ← DLARFB(A[k][k], T[k][k], A[k][n])
    FOR m = k+1..TILES-1
      A[k][n], A[m][n] ← DSSRFB(A[m][k], T[m][k], A[k][n], A[m][n])
```



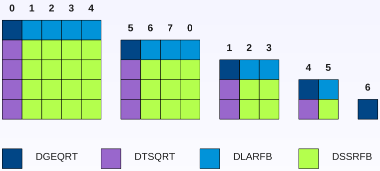


- ★ Different from the block algorithm.
- ★ Derived from out-of-core algorithm.
- ★ Input matrix stored and processed by square tiles.
- ★ Complex DAG.

Tile QR Factorization - Static pipeline

```
void dgeqrt(double *RV1, double *T);
void dtsqrt(double *R, double *V2, double *T);
void dlarfb(double *V1, double *T, double *C1);
void dssrfb(double *V2, double *T, double *C1, double *C2);

k = 0; n = my_core_id;
while (n >= TILES) {
   k++; n = n-TILES+k;
} m = k;

while (k < TILES && n < TILES) {
   next_n = n; next_m = m; next_k = k;

   next_m++;
   if (next_m == TILES) {
      next_n += cores_num;
      while (next_n >= TILES && next_k < TILES) {
         next_k++; next_n = next_n-TILES+next_k;
      } next_m = next_k;
   }

   if (n == k) {
      if (m == k) {
         while(progress[k][k] != k-1);
         dgeqrt(A[k][k], T[k][k]);
         progress[k][k] = k;
      }
      else{
         while(progress[m][k] != k-1);
         dtsqrt(A[k][k], A[m][k], T[m][k]);
         progress[m][k] = k;
      }
   }
   else {
      if (m == k) {
         while(progress[k][k] != k);
         while(progress[k][n] != k-1);
         dlarfb(A[k][k], T[k][k], A[k][n]);
      }
      else {
         while(progress[m][k] != k);
         while(progress[m][n] != k-1);
         dssrfb(A[m][k], T[m][k], A[k][n], A[m][n]);
         progress[m][n] = k;
      }
   }
   n = next_n; m = next_m; k = next_k;
}
```

- Work partitioned in one dimension (by block-rows).
- Cyclic assignment of work across all steps of the factorization (pipelining of factorization steps).
- Process tracking by a global progress table.
- Stall on dependencies (busy waiting).



Outline

1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

Outline

Libraries

- LAPACK:
  - LAPACK 3.2 on Intel machine;
  - LAPACK 3.1.1 on IBM machine;
- SCALAPACK:
  - SCALAPACK 1.8.0;
- Vendor libraries:
  - Intel MKL 10.1;
  - IBM ESSL 4.3;
  - IBM PESSL 3.3;
- Tile algorithms:
  - PLASMA ;
  - TBLAS.

Libraries

- LAPACK:
  - LAPACK 3.2 on Intel machine;
  - LAPACK 3.1.1 on IBM machine;
- SCALAPACK:
  - SCALAPACK 1.8.0;
- Vendor libraries:
  - Intel MKL 10.1;
  - IBM ESSL 4.3;
  - IBM PESSL 3.3;
- Tile algorithms:
  - PLASMA ;
  - TBLAS.

Outline

1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

Intel Xeon - 16 cores machine

- Node:
  - quad-socket quad-core Intel64 processors (16 cores).
- Intel Xeon processor:
  - quad-core;
  - Frequency: 2,4 GHz.
- Theoretical peak:
  - 9.6 Gflop/s/core;
  - 153.6 Gflop/s/node.
- System and compilers:
  - Linux 2.6.25;
  - Intel Compilers 11.0.

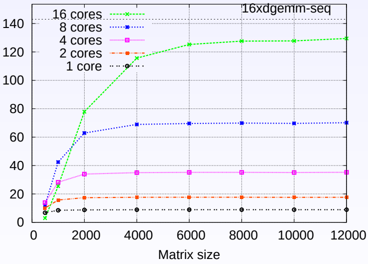
IBM Power6 - 32 cores machine

- Node:
  - 16 dual-core Power6 processors (32 cores).
- Power6 processor:
  - dual-core;
  - each core 2-way SMT;
  - L1: 64kB data + 64 kB instructions;
  - L2: 4 MB per core, accessible by the other core;
  - L3: 32 MB per processor, one controller per core (80 MB/s).
  - Frequency: 4,7 GHz.
- Theoretical peak:
  - 18.8 Gflop/s/core;
  - 601.6 Gflop/s/node.
- System and compilers:
  - AIX 5.3;
  - xlf version 12.1;
  - xlc version 10.1.

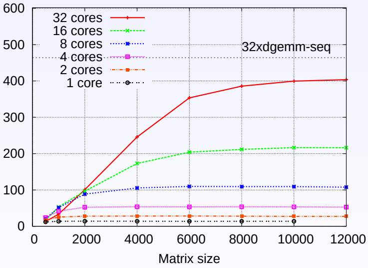
Outline

Performance metrics (How to read the graphs)

- Performance: Gflop/s (y-axis).
- Plots scaled to the theoretical peak.
- Parallel `DGEMM`.
- Upper bound: embarrassingly parallel fastest core kernel:
  - `DPOTRF` $(LL^T) \longrightarrow$ dgemm;
  - `DGEQRF` $(QR) \longrightarrow$ dssrfb;
  - `DGETRF` $(LU) \longrightarrow$ dssssm.



`Intel64- DGEMM`



`Power6- DGEMM`

Outline

Outline

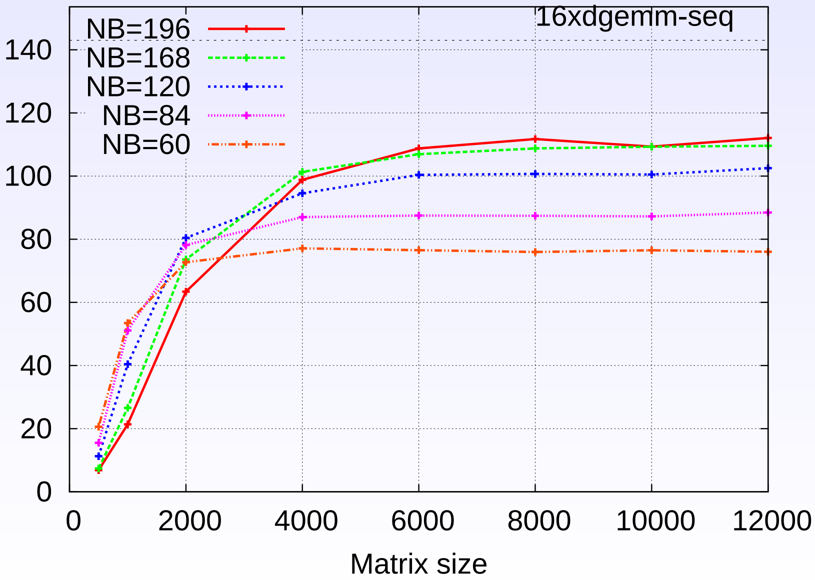
1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

Degrees of freedom

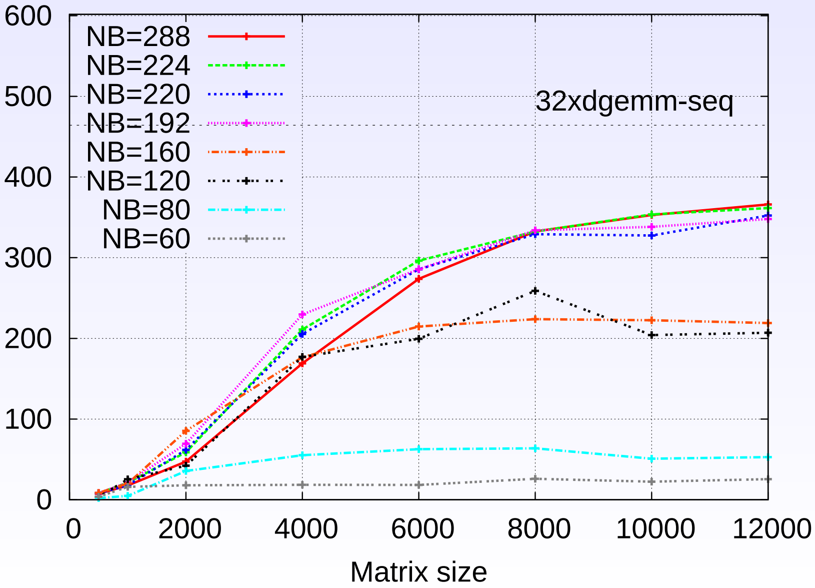
Input parameters of the serial core kernels:

- NB: tile size;
- IB: internal blocking (for dssrfb and dssssm only).

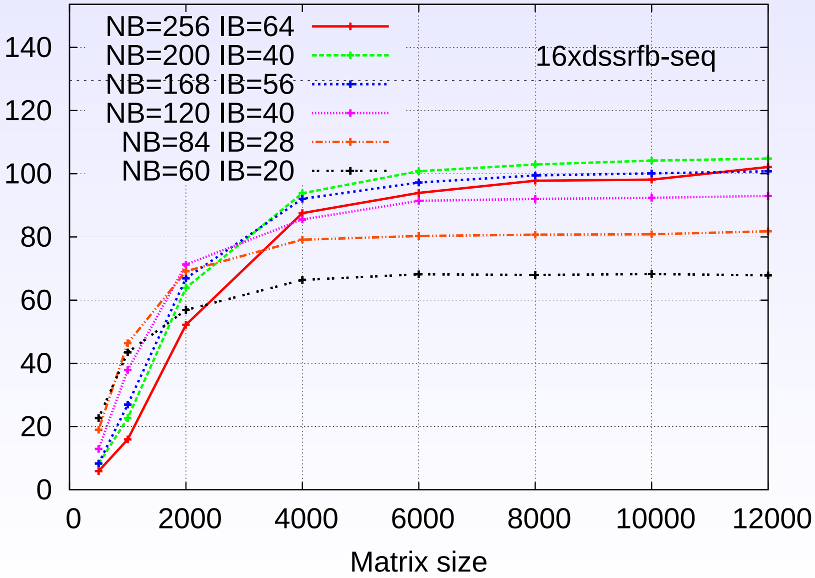
Impact of NB - DPOTRF- Intel64- 16 cores



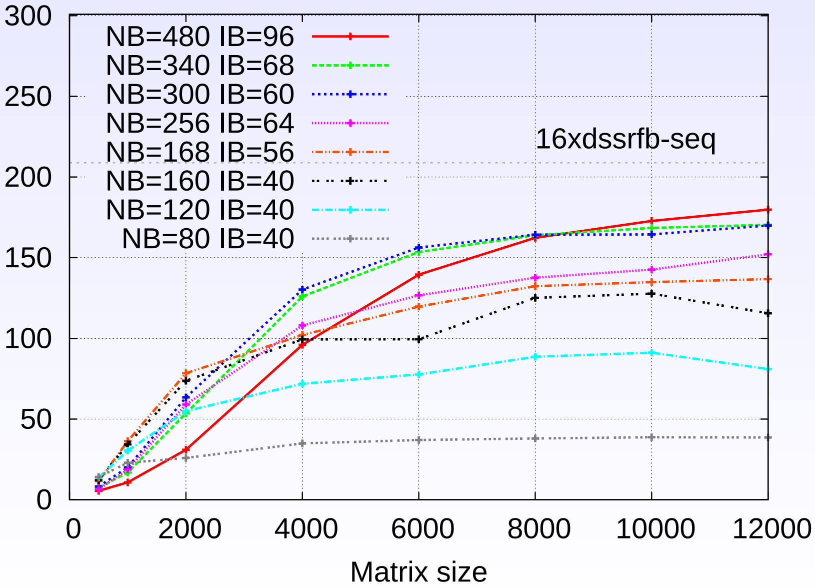
Impact of NB - DPOTRF- Power6- 32 cores



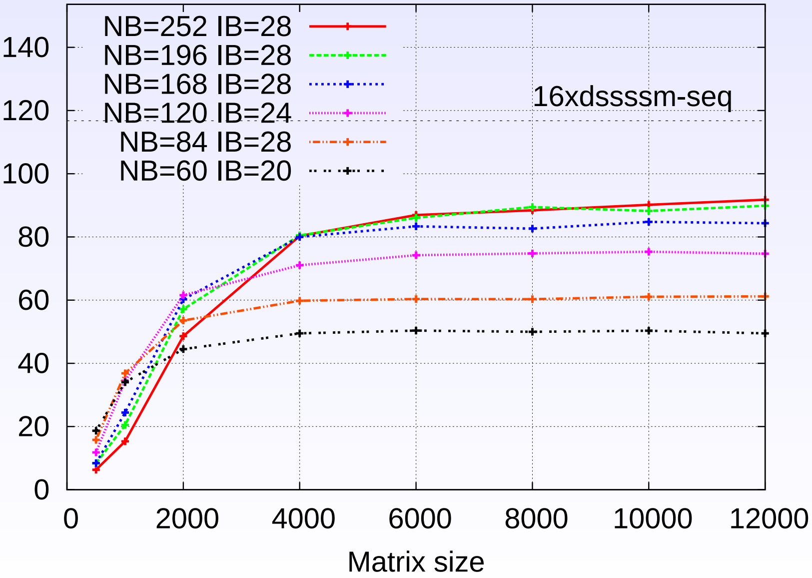
Impact of NB/IB - DGEQRF- Intel64- 16 cores



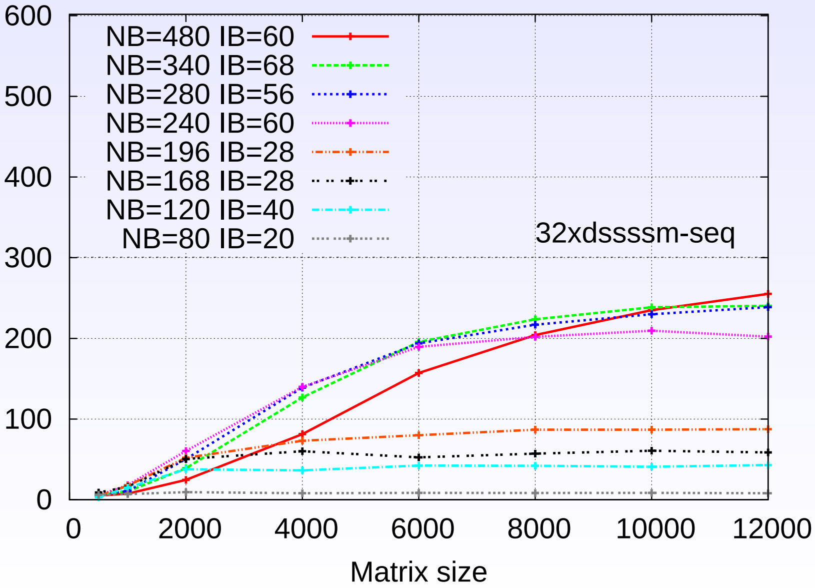
Impact of NB/IB - DGEQRF- Power6- 32 cores



Impact of NB/IB - DGETRF- Intel64- 16 cores



Impact of NB/IB - DGETRF- Power6- 32 cores



Exhaustive search

For "each" matrix size and number of cores:

1. Time PLASMA on all NB/IB samples;
2. Select the best sample.

Number of samples

- ⋆ $|\{(IB, NB) \mid IB|NB, 40 \leq NB \leq 500, 4 \leq IB \leq NB\}|$=1352;
- ⋆ all combinations cannot be explored on large executions;

$\longrightarrow$ need for a pruned search.

Exhaustive search

For ”each” matrix size and number of cores:

1. Time PLASMA on all NB/IB samples;
2. Select the best sample.

Number of samples

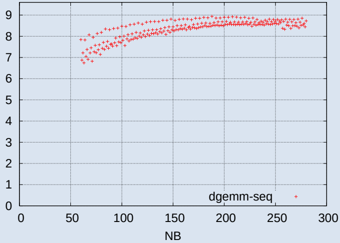
- ⋆ $|\{(IB, NB) \mid IB|NB, 40 \leq NB \leq 500, 4 \leq IB \leq NB\}|$=1352;
- ⋆ all combinations cannot be explored on large executions;

$\longrightarrow$ need for a pruned search.

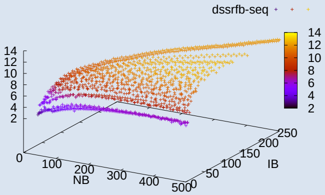
Pruned search

Method

1. Time serial core kernels (dgemm, dssrfb, dssssm).



Intel64 - dgemm



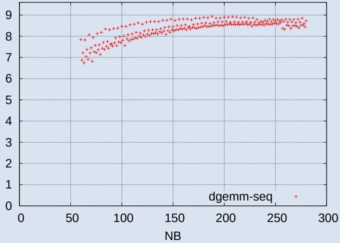
Power6 - dssrfb

2. Pick up the "best" NB or NB/IB samples (pruning);
3. Select one per matrix size and number of cores.

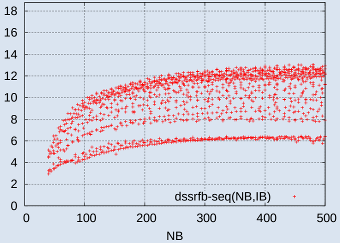
Pruned search

Method

1. Time serial core kernels (dgemm, dssrfb, dssssm).



Intel64 - dgemm



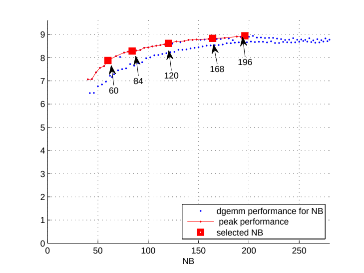
Power6 - dssrfb

2. Pick up the "best" NB or NB/IB samples (pruning);
3. Select one per matrix size and number of cores.

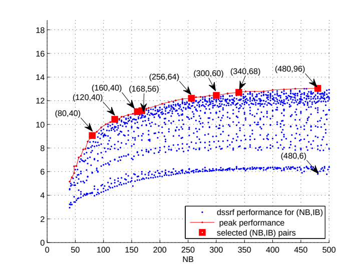
Pruned search

Method

1. Time serial core kernels (dgemm, dssrfb, dssssm).



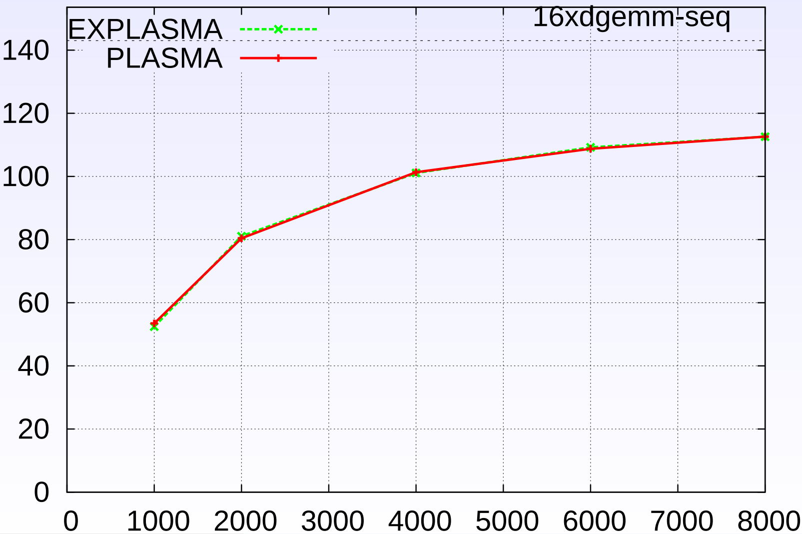
Intel64 - dgemm



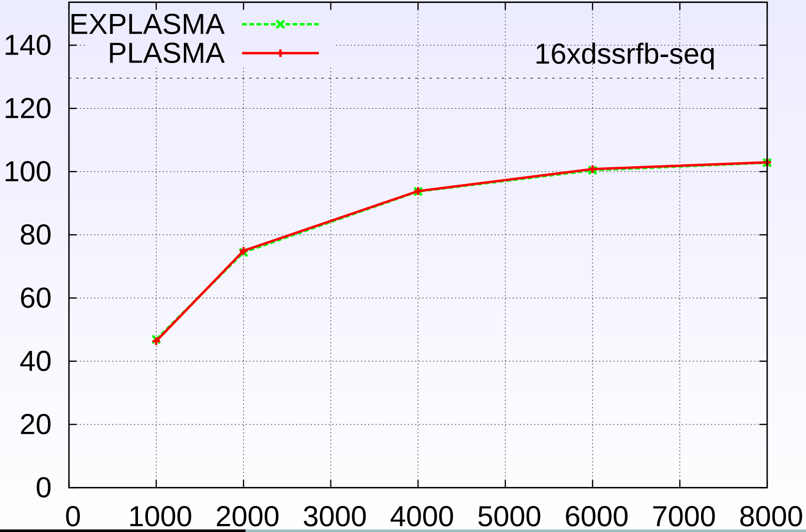
Power6 - dssrfb

2. Pick up the "best" NB or NB/IB samples (pruning);
3. Select one per matrix size and number of cores.

Exhaustive search VS pruned search
Intel64- 16 cores - DPOTRF

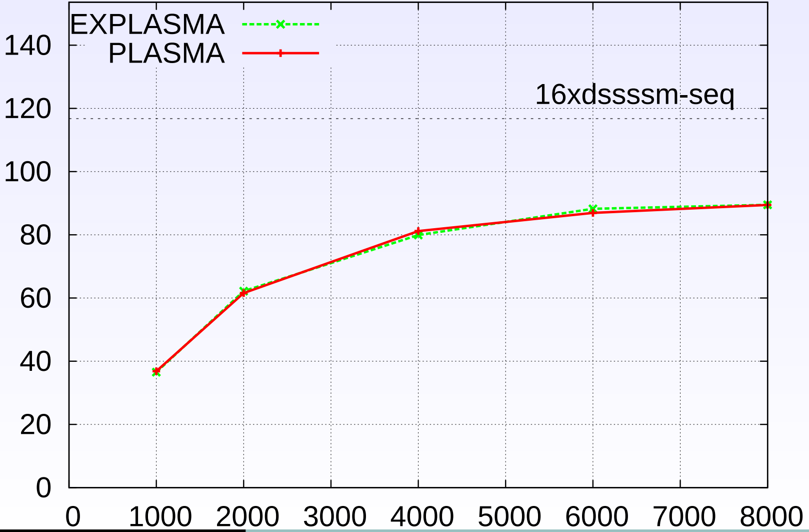


Exhaustive search VS pruned search
Intel64- 16 cores - DGEQRF



Exhaustive search VS pruned search

Intel64- 16 cores - DGETRF



Other software

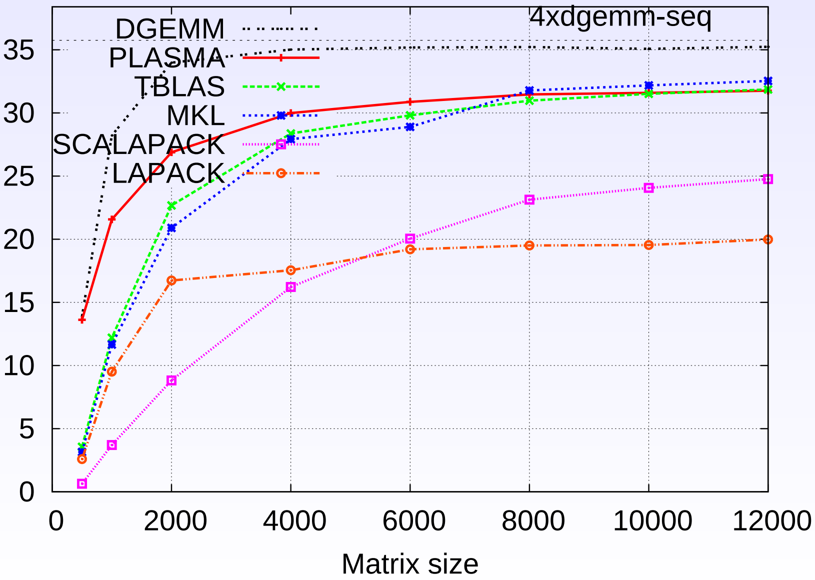
- ⋆ PLASMA: pruned search.
- ⋆ TBLAS: exhaustive search.
- ⋆ SCALAPACK, PESSL: exhaustive search.
- ⋆ LAPACK, MKL, ESSL: tuned by vendor.

Outline

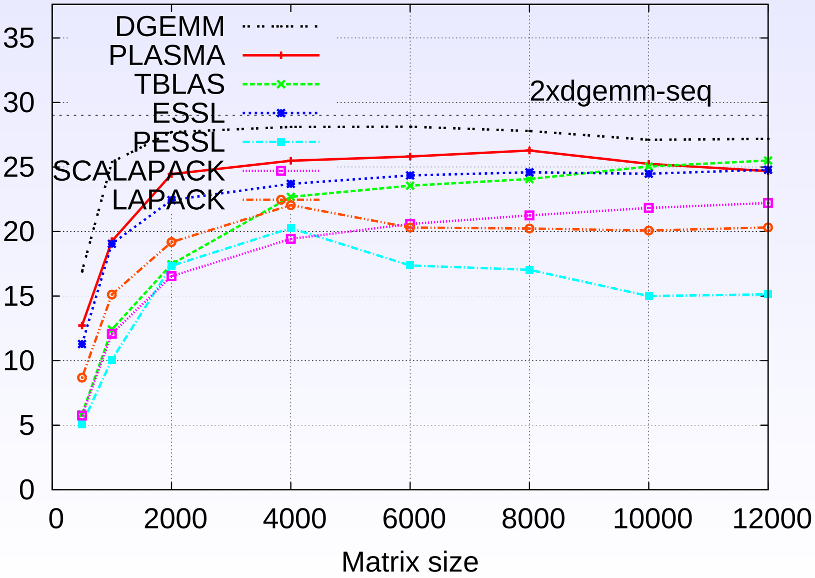
1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

Outline

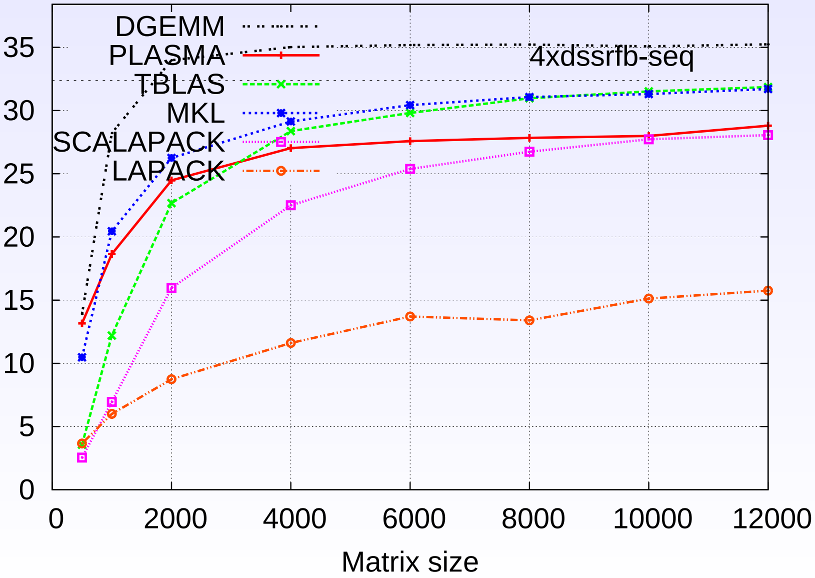
DPOTRF- Intel64- 4 cores



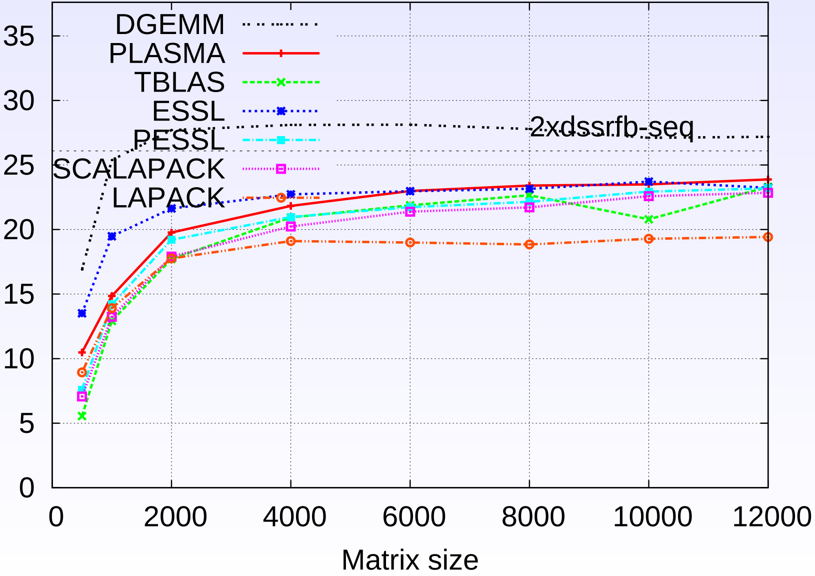
DPOTRF- Power6- 2 cores



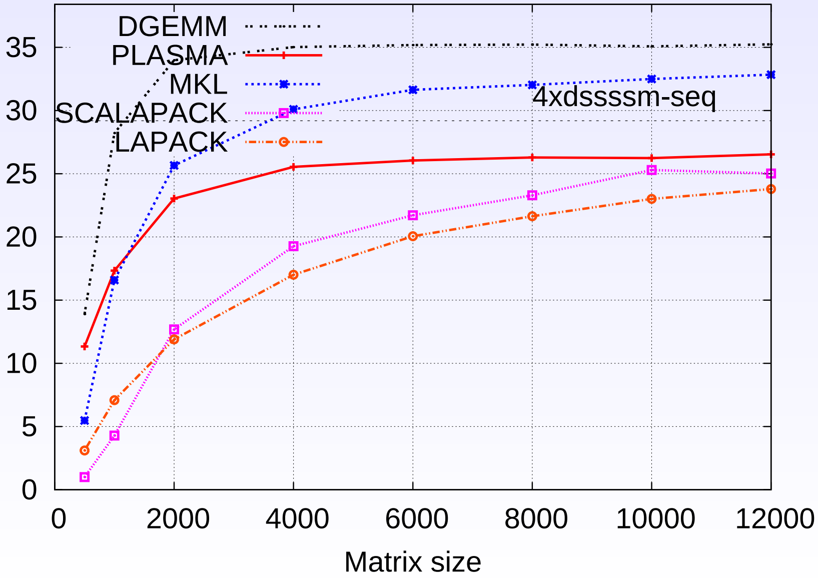
DGEQRF- Intel64- 4 cores



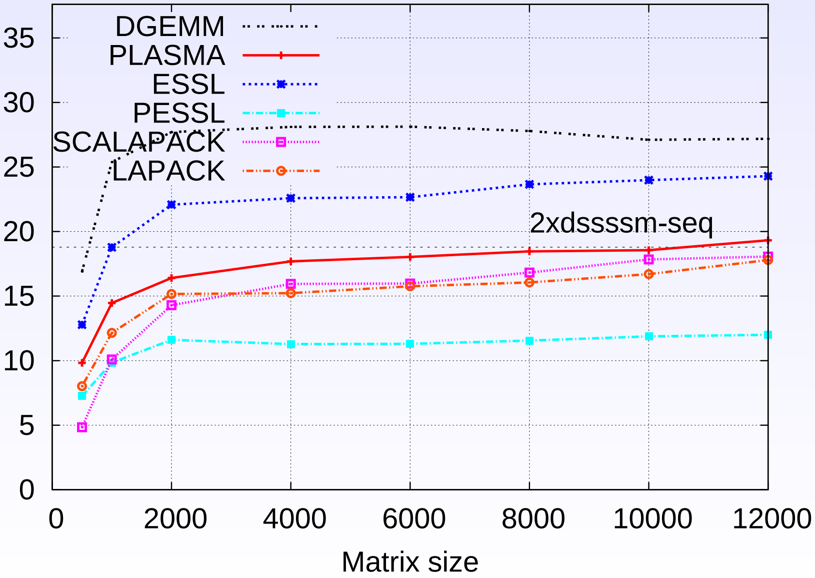
DGEQRF- Power6- 2 cores



DGETRF- Intel64- 4 cores



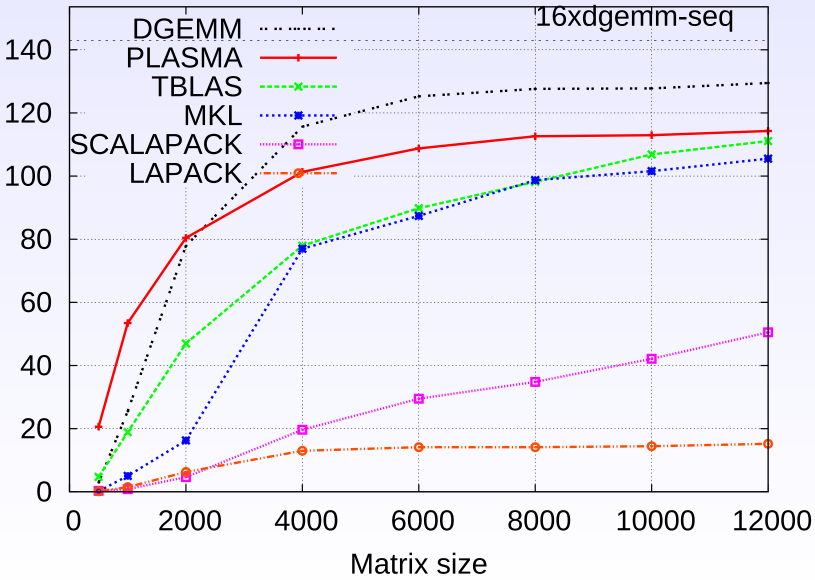
DGETRF- Power6- 2 cores



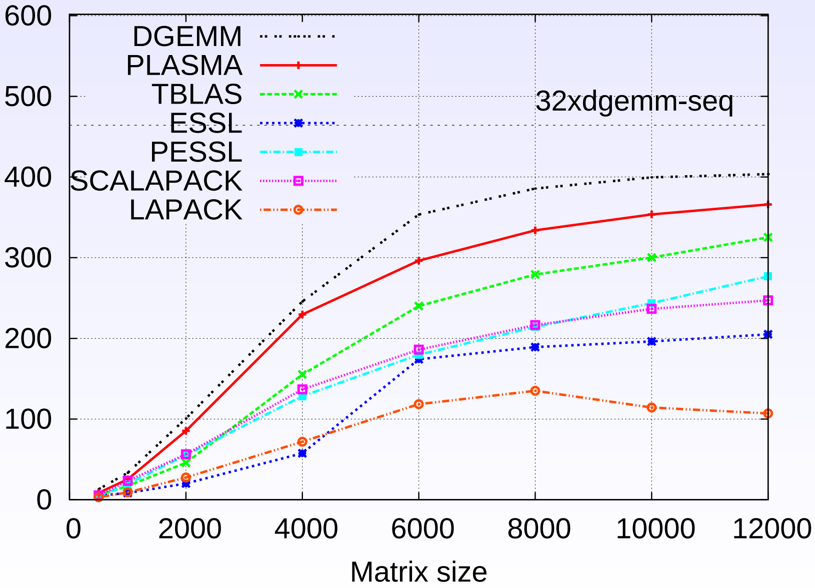
Outline

1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

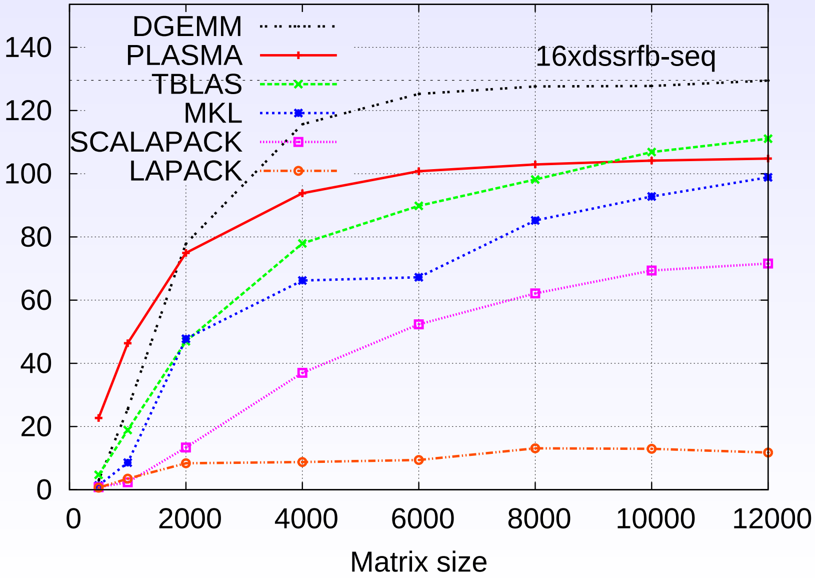
DPOTRF- Intel64- 16 cores



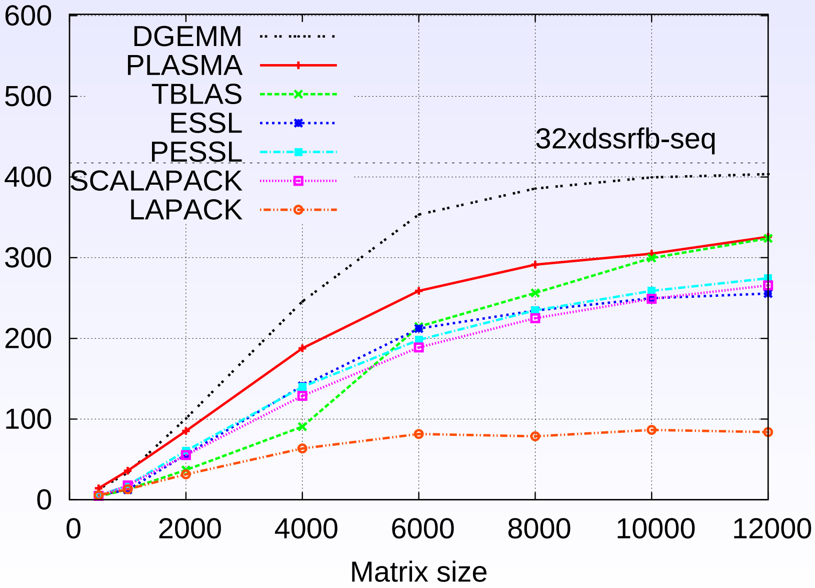
DPOTRF- Power6- 32 cores



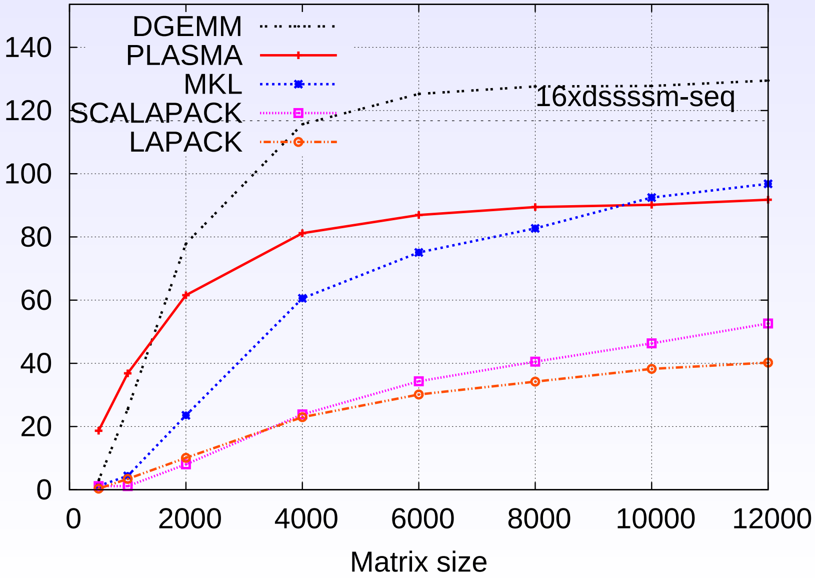
DGEQRF- Intel64- 16 cores



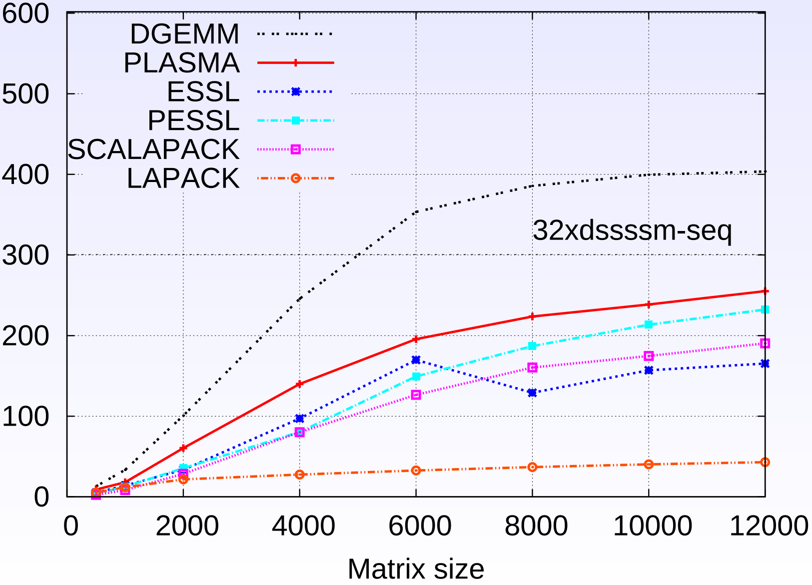
DGEQRF- Power6- 32 cores



DGETRF- Intel64- 16 cores



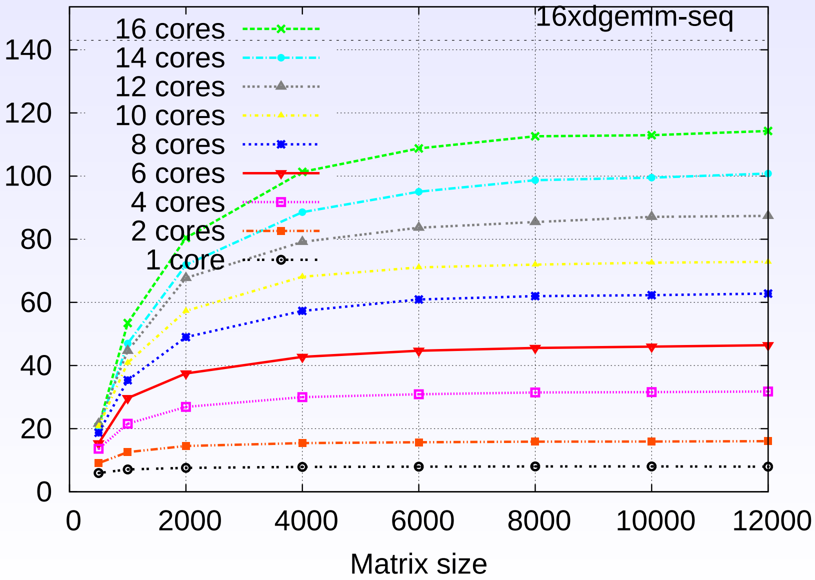
DGETRF- Power6- 32 cores



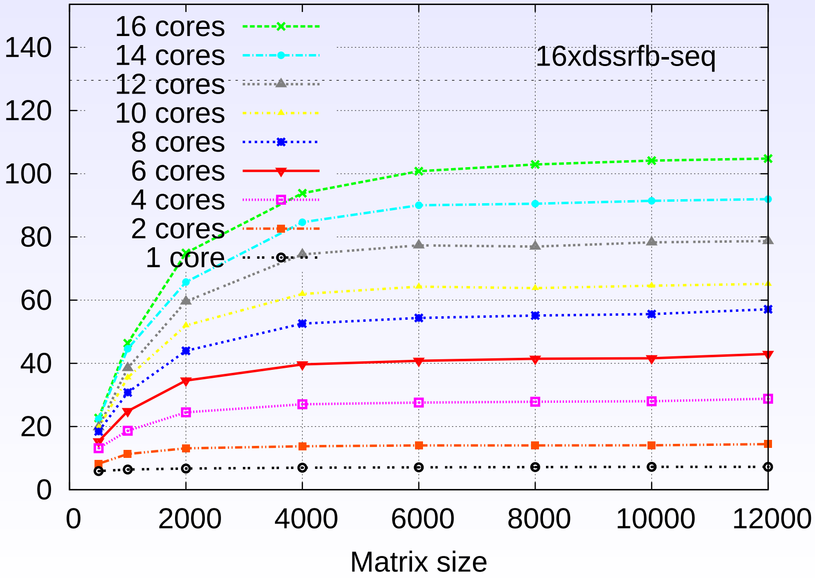
Outline

1. Tile Algorithms
   - Cholesky Factorization
   - QR (&LU) Factorizations
2. Experimental environment
   - Libraries
   - Hardware
   - Metrics
3. Tuning
   - PLASMA
4. Comparison against other libraries
   - Experiments on few cores
   - Experiments on a large number of cores
   - PLASMA scalability
5. Conclusion and current work

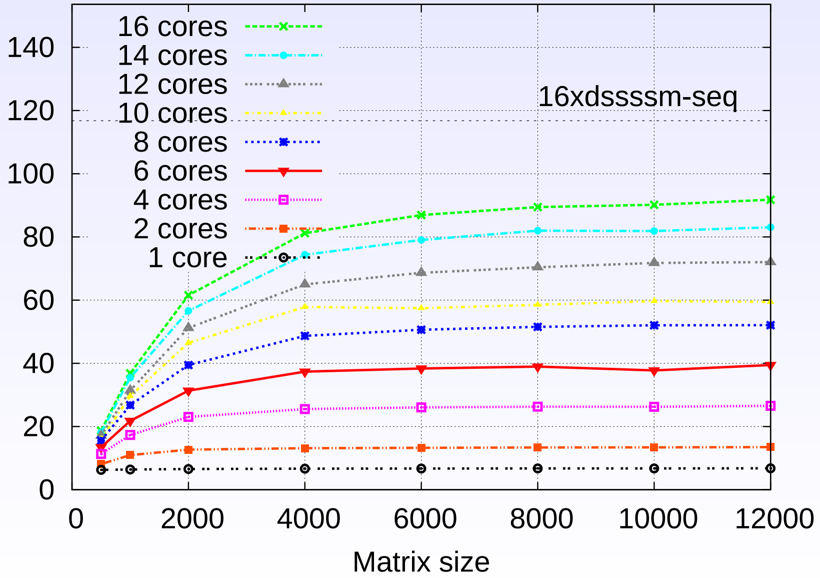
PLASMA- DPOTRF- Intel64



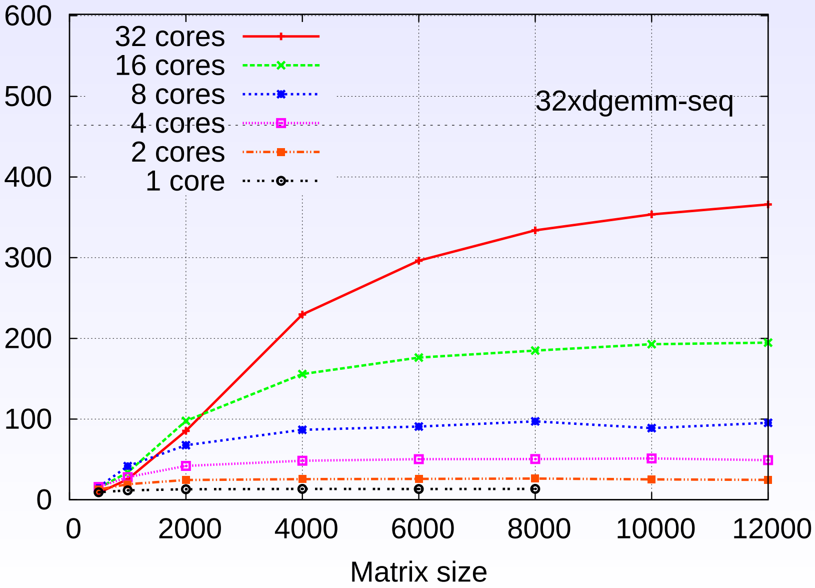
PLASMA- DGEQRF- Intel64



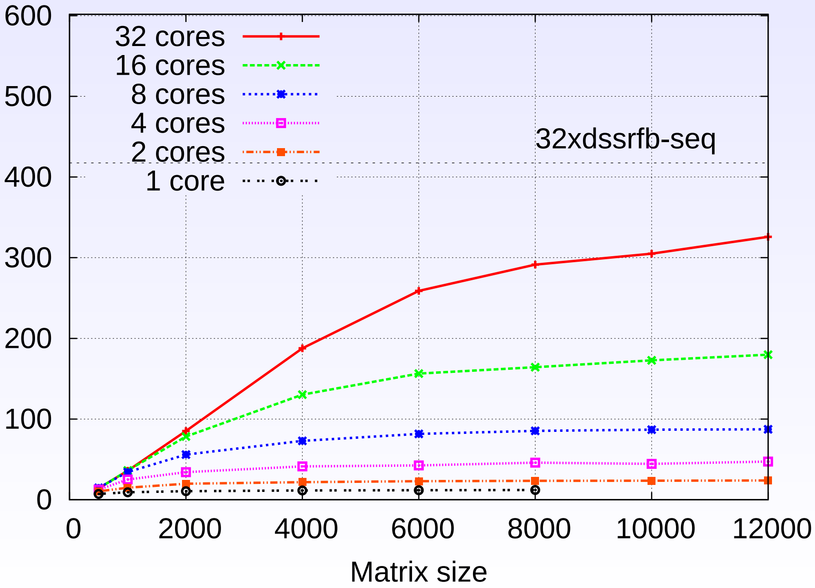
PLASMA- DGETRF- Intel64



PLASMA- DPOTRF- Power6



PLASMA- DGEQRF- Power6



Outline

Conclusion

- ★ Performance brought by tile algorithms:
  - ☹ Possible overheads:
    - extra-flops;
    - kernels not optimized.
  - ☺ Benefits:
    - better data reuse;
    - better scheduling opportunities.
- ★ Better scalability.
- ★ Importance of tuning:
  - $\longrightarrow$ efficient pruned search.

Current work

- ⋆ Compute-intensive kernels:
  successive BLAS-3 calls $\longrightarrow$ single BLAS-3 call.
- ⋆ Dynamic scheduling:
  $\longrightarrow$ Piotr's presentation.
- ⋆ Improve scalability for small matrix sizes:
  $\longrightarrow$ increase parallelism (tile TSQR).
- ⋆ Generalization to other linear algebra algorithms:
  $\longrightarrow$ two-sided factorizations.

Thanks

Questions?

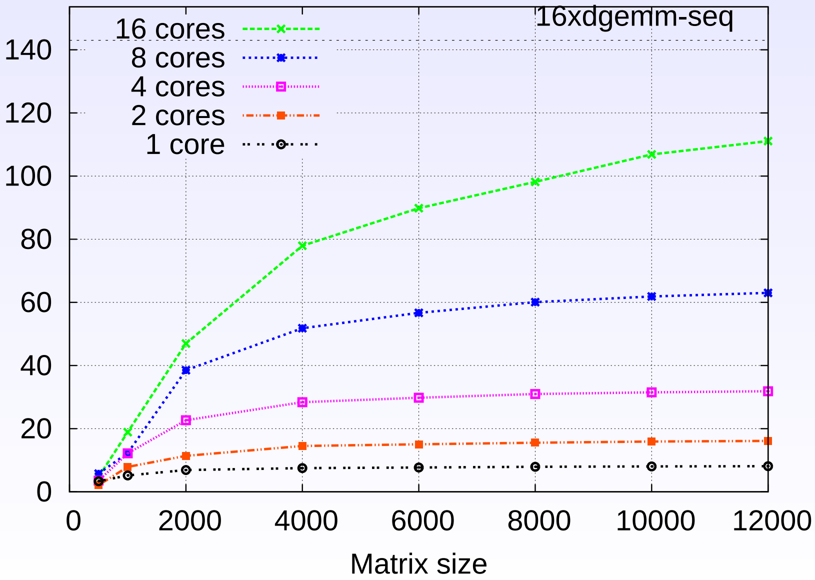
Outline

1. Scalability of other libraries
   - TBLAS
   - MKL- ESSL
   - SCALAPACK- PESSL
   - LAPACK

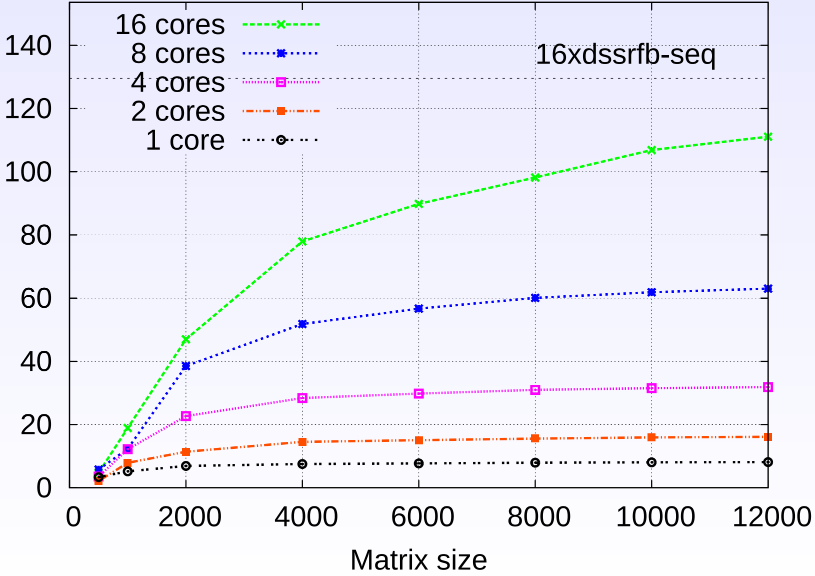
Outline

1. Scalability of other libraries
   - TBLAS
   - MKL- ESSL
   - SCALAPACK- PESSL
   - LAPACK

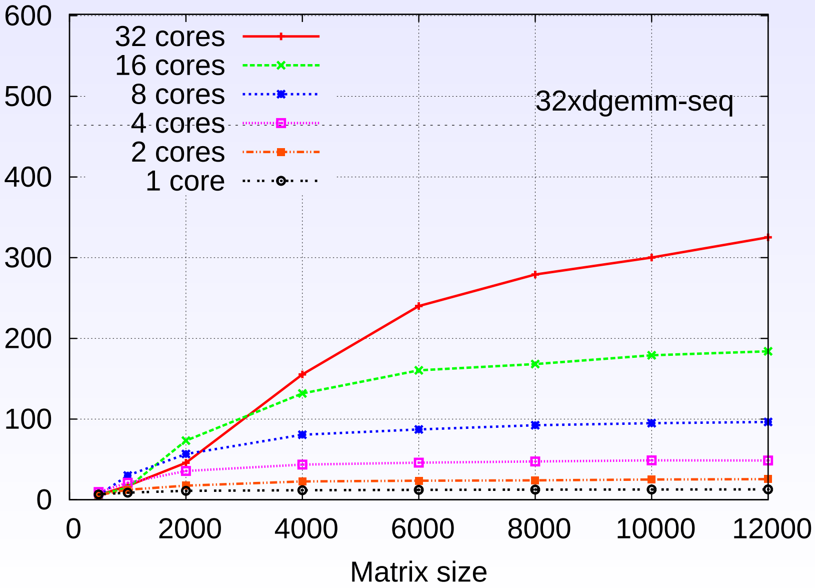
TBLAS- DPOTRF- Intel64



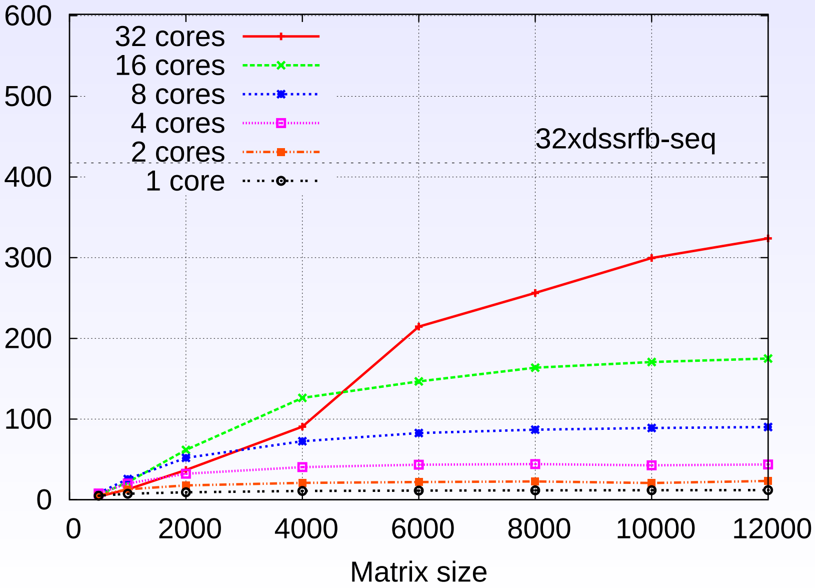
TBLAS- DGEQRF- Intel64



TBLAS- DPOTRF- Power6



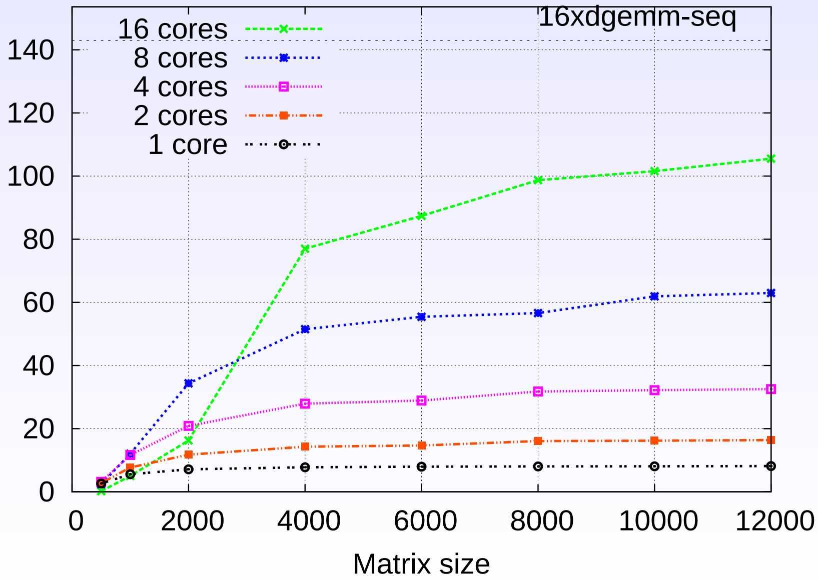
TBLAS- DGEQRF- Power6



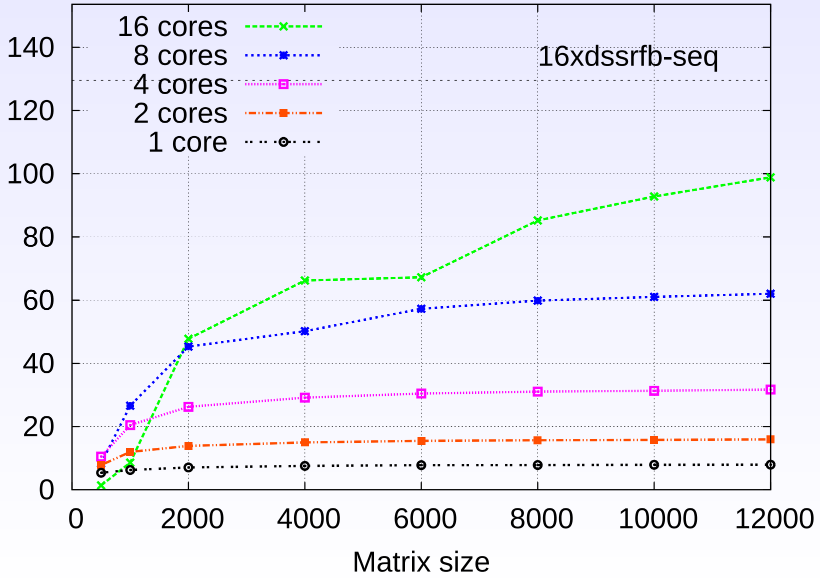
Outline

1. Scalability of other libraries
   - TBLAS
   - MKL- ESSL
   - SCALAPACK- PESSL
   - LAPACK

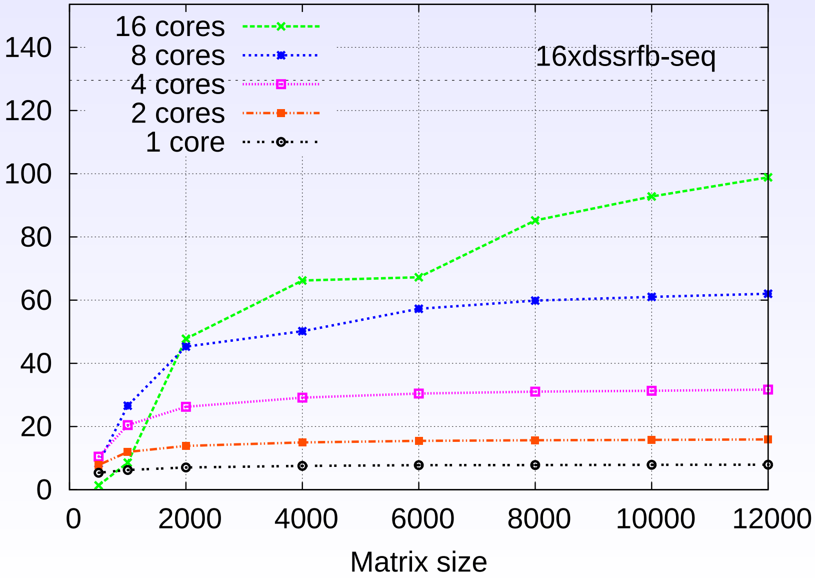
MKL- DPOTRF- Intel64



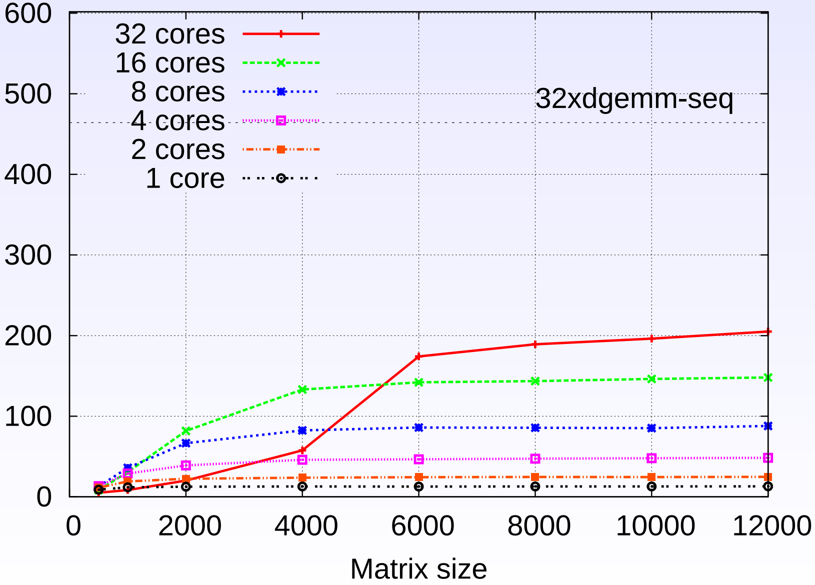
MKL- DGEQRF- Intel64



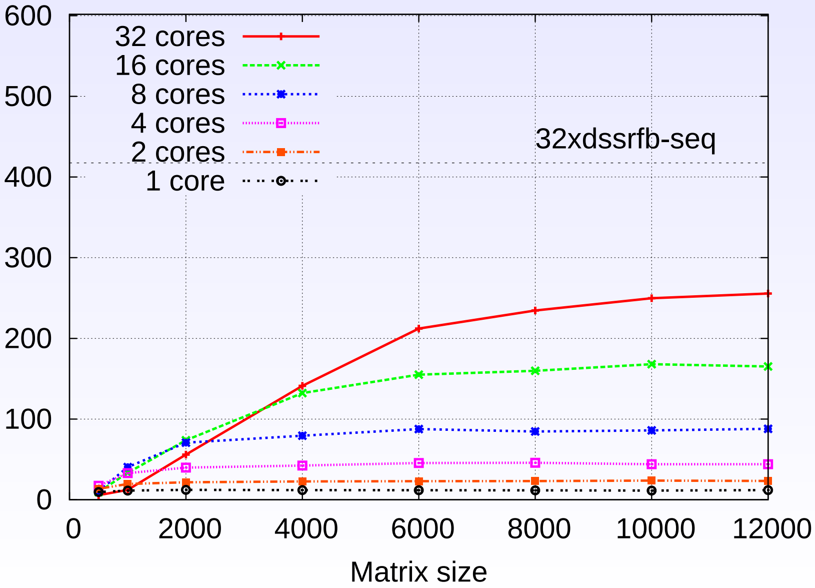
MKL- DGETRF- Intel64



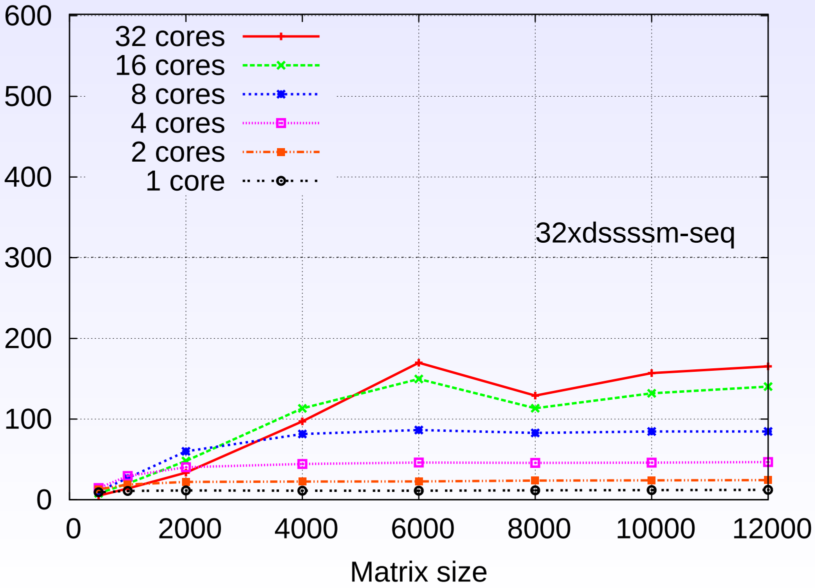
ESSL- DPOTRF- Power6



ESSL- DGEQRF- Power6



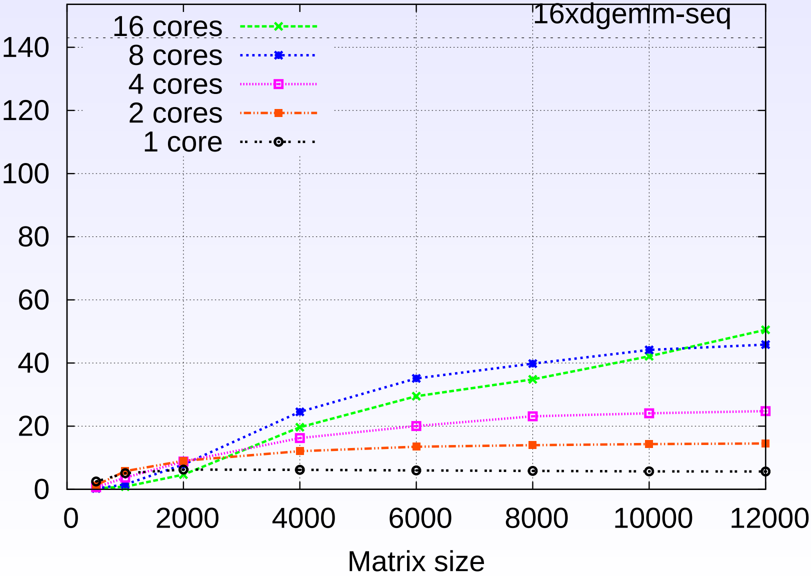
ESSL- DGETRF- Power6



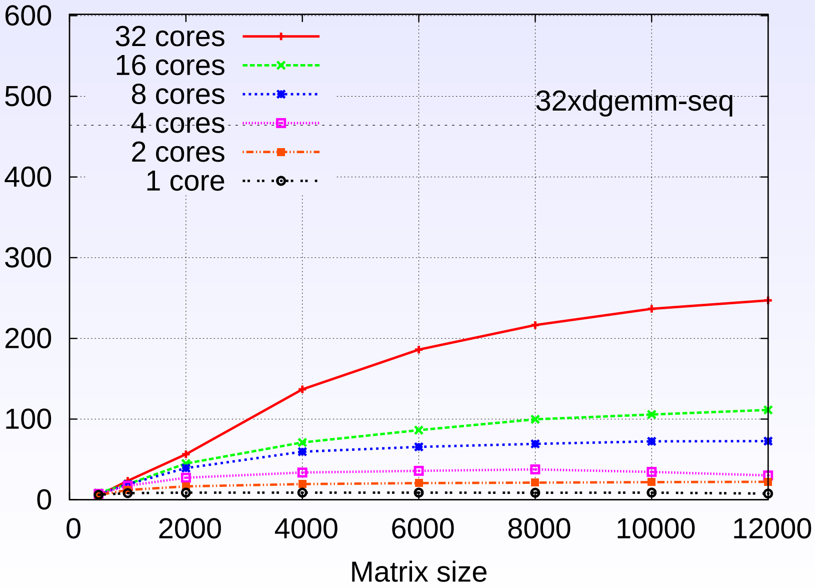
Outline

1. Scalability of other libraries
   - TBLAS
   - MKL- ESSL
   - SCALAPACK- PESSL
   - LAPACK

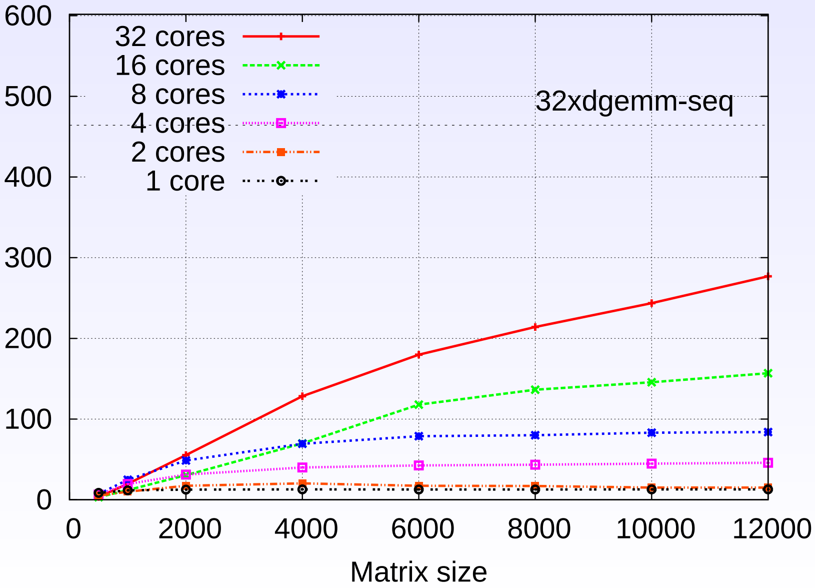
SCALAPACK- DPOTRF- Intel64



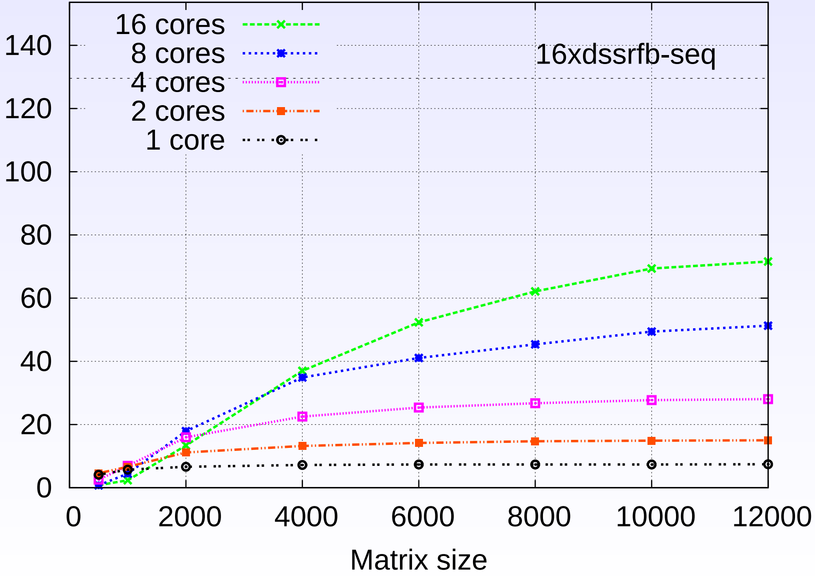
SCALAPACK- DPOTRF- Power6



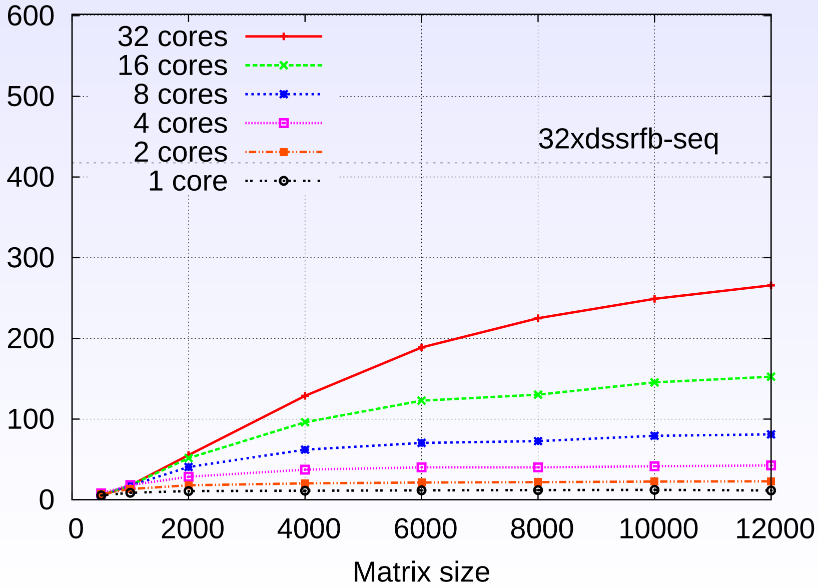
PESSL- DPOTRF- Power6



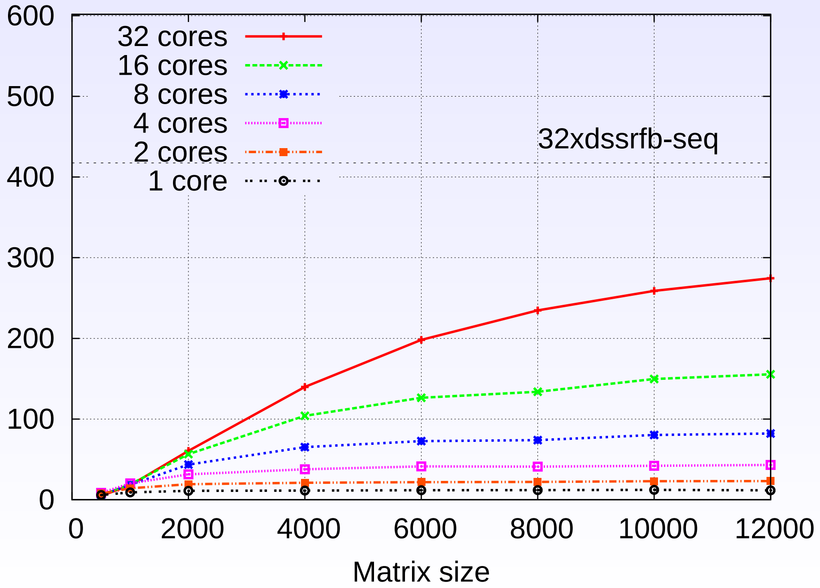
SCALAPACK- DGEQRF- Intel64



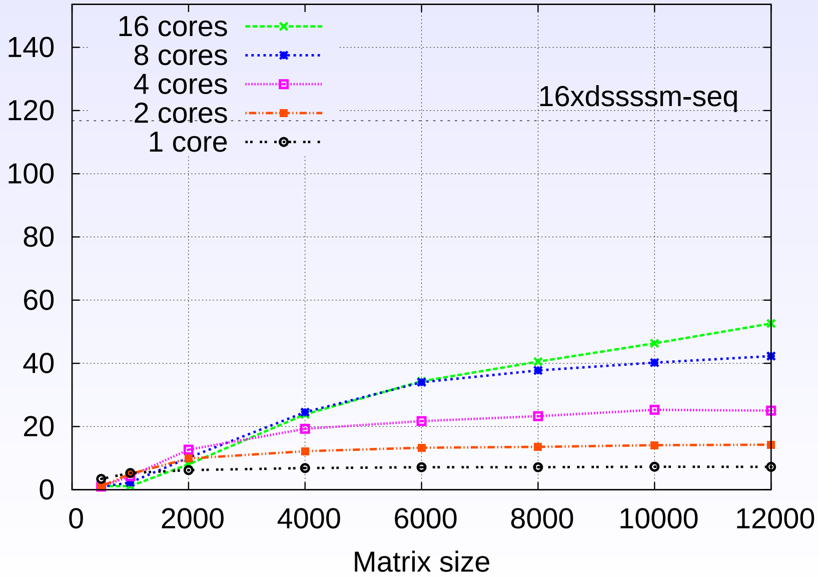
SCALAPACK- DGETRF- Power6



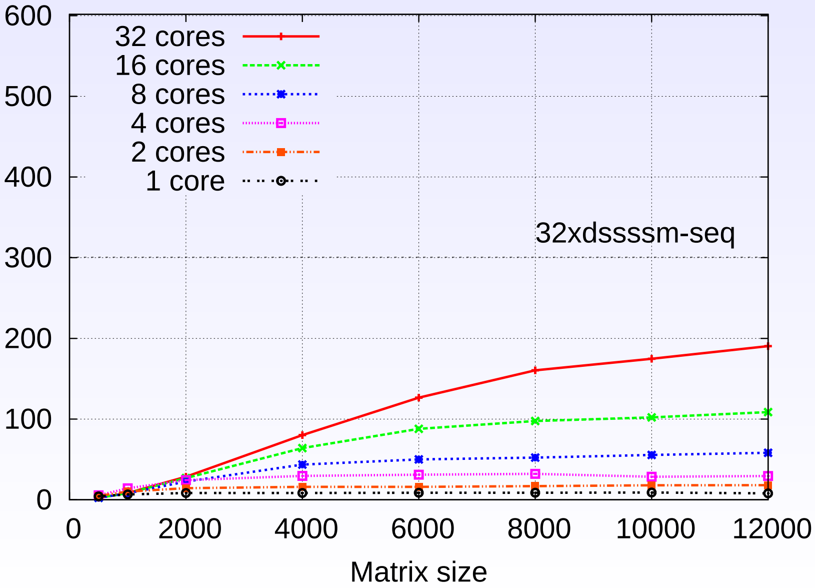
PESSL- DGEQRF- Power6



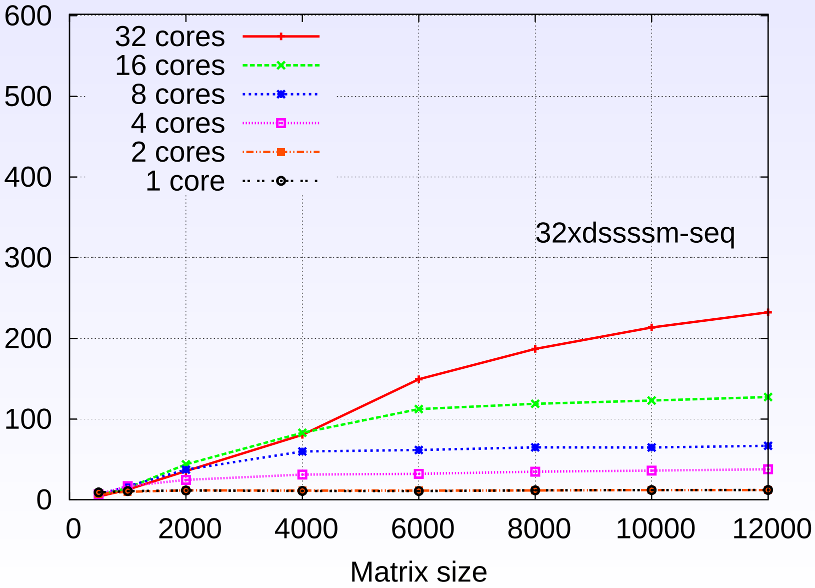
SCALAPACK- DGETRF- Intel64



SCALAPACK- DGETRF- Power6



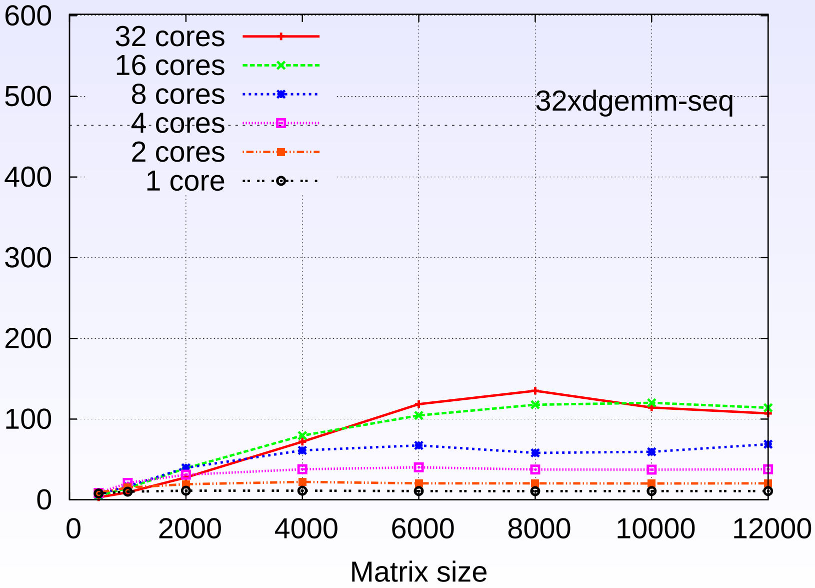
PESSL- DGETRF- Power6



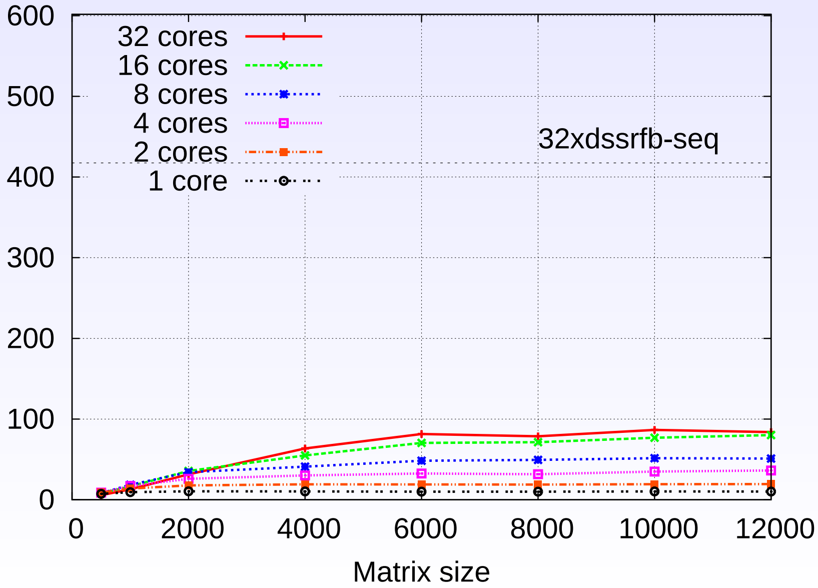
Outline

1. Scalability of other libraries
   - TBLAS
   - MKL- ESSL
   - SCALAPACK- PESSL
   - LAPACK

LAPACK- DPOTRF



LAPACK- DGEQRF



LAPACK- DGETRF