Building a High-Performance Solver Stack on Top of a Runtime System (Part 2/2)

Journée scientifique PlaFRIM - 10 ans

Olivier Aumage & Mathieu Faverge
High-Performance Solver Stack

**Solver Stack**

- Team SATANAS / HiePACS
- High performance numerical libraries
  - Task parallelism
  - Target task-based runtimes
- Heterogeneous, accelerated archs.

**STARPU Runtime System**

- Team SATANAS / Storm
- Task scheduling
  - Performance modeling
  - Data management
- Heterogeneous, accelerated archs.

---

HPC Applications

- **CHAMELEON**
  - Dense linear algébra
- **PASTIX**
  - Sparse linear algebra
- **SCALFMM**
  - Fast multipole method

Drivers (CPU, CUDA, OpenCL, ...)

- **StarPU** Runtime System

- **CPU**
- **GPU**
- **...**
Different scale/problem, different solver

MAPHYs
- Hybrid solver (Direct + Iterative)
- Domain decomposition

PaStiX
- Sparse direct solver
- Supernodal method
- Runtime systems (STARPU, PARSEC)
- Heterogeneous (distributed) architectures

Chameleon
- Dense linear algebra
- Replacement to SCALAPACK
- Tiled algorithms
- Runtime systems (STARPU, QUARK, PARSEC, OPENMP, …)
- Heterogeneous distributed architectures
What results with Plafrim on the last 10 years?

**CHAMELEON/ DLA**
1. Distributed QR factorizations and SVD problem
2. QDWH: Alternative solution to compute the SVD

**PaStiX/ SLA**
1. Heterogeneous sparse direct solver
2. Low-rank sparse direct solver
1

Dense linear algebra
DPLASMA: Singular Value Decomposition

\[ A = U \Sigma V^T \]

- Collaboration with Univ. of Colorado, ICL and Inria ROMA
- Focus on getting the singular values only (GEVAL)
- Use three steps algorithms:
  - **GE2BND**: Reduce the general matrix to general band
  - **BND2BD**: Reduce the general band to bidiagonal
  - **BD2VAL**: Compute the singular values from the bidiagonal
DPLASMA: Singular Value Decomposition

\[ A = U \Sigma V^T \]

- Collaboration with Univ. of Colorado, ICL and Inria ROMA
- Focus on getting the singular values only (GEVAL)
- Use three steps algorithms:
  
  **GE2BND**  Reduce the general matrix to general band
  **BND2BD**  Reduce the general band to bidiagonal
  **BD2VAL**  Compute the singular values from the bidiagonal
DPLASMA: Singular Value Decomposition

\[ A = U \Sigma V^T \]

- Collaboration with Univ. of Colorado, ICL and Inria ROMA
- Focus on getting the singular values only (GEVAL)
- Use three steps algorithms:
  - **GE2BND**: Reduce the general matrix to general band
  - **BND2BD**: Reduce the general band to bidiagonal (PLASMA)
  - **BD2VAL**: Compute the singular values from the bidiagonal
DPLASMA: Singular Value Decomposition

\[ A = U \Sigma V^T \]

- Collaboration with Univ. of Colorado, ICL and Inria ROMA
- Focus on getting the singular values only (GEVAL)
- Use three steps algorithms:
  - **GE2BND** Reduce the general matrix to general band
  - **BND2BD** Reduce the general band to bidiagonal (PLASMA)
  - **BD2VAL** Compute the singular values from the bidiagonal (MKL)
Exploit the flexibility of the tile QR algorithm

**TS** Triangle on top of square
- more data locality 😊
- efficient implementation 😊
- only a single killer per panel 😞

**TT** Triangle on top of triangle
- several killers per panel 😊
- more parallelism 😊
- less efficient implementation 😞
Shared Memory - Tall and Skinny case

- $N = 2000$
- $q = 13 \approx \frac{\text{nbcores}}{2}$
- BI\text{DIAG}FLATTS lacks of parallelism
- R-BI\text{DIAG} is quickly beneficial

- $N = 10000$
- $q = 63 \approx \frac{5}{2} \text{nbcores}$
- BI\text{DIAG}FLATTS has a better efficiency than TT trees
- R-BI\text{DIAG} is quickly beneficial
Distributed Memory - Strong scaling - T&S

- $N = 2\,000$
- $M = 2\,000\,000$
- $q = 13 \approx \frac{nbcores}{2}$
- AUTO gives a 25% improvement
- $\approx 10$ times faster on GEVAL

- $N = 10\,000$
- $M = 1\,000\,000$
- $q = 63 \approx \frac{5}{2} nbcores$
- AUTO gives a 25% improvement
- $\approx 3$ times faster on GEVAL

GE2BND

GE2VAL
Distributed Memory - Weak scaling - T&S

- $N = 2000$
- $M = 80\,000\,nb_{nodes}$
- $q = 13 \approx \frac{nb_{cores}}{2}$
- AUTO gives a 25% improvement
- $\approx 10$ times faster on GEVAL

- $N = 10\,000$
- $M = 100\,000\,nb_{nodes}$
- $q = 63 \approx \frac{5}{2} nb_{cores}$
- AUTO gives a 25% improvement
- $\approx 2.5$ times faster on GEVAL

---

GE2BND

GE2VAL
### QDWH: Polar Decomposition

#### Context
- Collaboration with KAUST
- Study alternative algorithms to compute singular and eigenvalues (QDWH, Zolo).
- More costly than classical SVD (up to $90n^3$ for QDWH), but more scalable

#### Objective
- Exploit CHAMELEON and STARPU
- Exploit task-based implementation to remove synchronizations
- Propose an efficient solution across platforms

```c
/* Compute the polar decomposition $A = U P H$ using QDWH */
while Not converged do
    dlacpy (U, U1) ▷ Backup $U_{k-1}$

    /* Compute $U_k$ from $U_{k-1}$ */
    if $c > 100$ then
        C = [$C_1$ $C_2$] = [$\sqrt{c}U_{k-1}$ $I$]
       dgeqrf (C) ▷ $C = QR = [Q_1 | Q_2] R$
        dorgqr (C) ▷ $C = Q = [Q_1 | Q_2]$
        dgemm (Q_1, Q_2^T, U) ▷
        $U_k = \frac{1}{\sqrt{c}} (a - \frac{b}{c}) Q_1 Q_2^T + \frac{b}{c} U_{k-1}$
    else
        dlaset (Z, 0., 1.) ▷ $Z = I$
        dgemm (U^T, U, Z) ▷ $Z_k = I - c U_{k-1}^T U_{k-1}$
        dgeadd (U, B) ▷ $B = U_{k-1}^T$
        dposv (Z, B) ▷ Solve $Z_k x = U_{k-1}^T$
        dgeadd (B, U) ▷
        $U_k = \frac{b}{c} U_{k-1} + (a - \frac{b}{c}) (U_{k-1} W_{k-1}^{-1}) W_{k-1}$
    end if
    dgeadd (U, U1) ▷ $U_k - U_{k-1}$
    dlange (U1, conv) ▷ $conv = \|U_k - U_{k-1}\|_F$

    $k = k + 1$
end while
```
QDWH: Polar Decomposition

Context

- Collaboration with KAUST
- Study alternative algorithms to compute singular and eigen values (QDWH, Zolo).
- More costly than classical SVD (up to $90n^3$ for QDWH), but more scalable

Objective

- Exploit CHAMELEON and STARPU
- Exploit task-based implementation to remove synchronizations
- Propose an efficient solution across platforms

```c
/* Compute the polar decomposition $A = UPH$ using QDWH */
while Not converged do
  dlacpy_Async(U, U1)  // Backup $U_{k-1}$

  /* Compute $U_k$ from $U_{k-1}$ */
  if $c > 100$ then
    $C = \begin{bmatrix} C_1 \n C_2 \end{bmatrix}$ = $\begin{bmatrix} \sqrt{c}U_{k-1} \n I \end{bmatrix}$
    dgeqrf_Async(C)  // $C = QR = \begin{bmatrix} Q_1 \n Q_2 \end{bmatrix} R$
    dorgqr_Async(C)
    dgemm_Async(Q_1, Q_2^T, U)
    $U_k = \frac{1}{\sqrt{c}} \left( a - \frac{b}{c} \right) Q_1 Q_2^T + \frac{b}{c} U_{k-1}$
  else
    dlaset_Async(Z, 0., 1.)
    dgemm_Async(U^T, U, Z)
    dgeadd_Async(U, B)
    dposv_Async(Z, B)
    dgeadd_Async(B, U)
    $U_k = \frac{b}{c} U_{k-1} + \left( a - \frac{b}{c} \right) (U_{k-1} W_{k-1}^{-1}) W_{k-1}$
  end if
  dgeadd_Async(U, U1)  // $U_k - U_{k-1}$
  dlange_Async(U1, conv)
  RUNTIME.sequence.wait()
  $k = k + 1$
end while
```
Exploiting task-based runtime systems (Pipelining)

**Synchronous**
- Synchronizations at:
  - each step of QDWH
  - within the steps
- Too much idle time

**Asynchronous**
- Synchronizations are removed:
  - between the steps of QDWH
  - within the steps
- Idle time is reduced to minimum
Assessing the task-based QDWH scalability

Matrix size

GFlop/s

32 Threads
16 Threads
8 Threads
4 Threads

64 Threads
32 Threads
16 Threads
8 Threads
4 Threads

600
300
0
900
600

600
300
0
900
600

600
300
0
900
600

600
300
0
900
600

Haswell
Haswell + 8xK80
Broadwell

GFlop/s

200
100
0
600
400
200

600
400
200

600
400
200

600
400
200

CPU + 16 GPU
CPU + 8 GPU
CPU + 4 GPU
CPU + 2 GPU
CPU + 1 GPU
CPU

CPU + 16 GPU
CPU + 8 GPU
CPU + 4 GPU
CPU + 2 GPU
CPU + 1 GPU
CPU
Task-based QDWH performance across various architectures

![Graph showing performance across various architectures](image-url)
Sparse linear algebra
Heterogeneous architectures: from dense to sparse? From CHAMELEON to PaStiX?

Context
- PaStiX sparse direct solver
- Task-based implementation (X. Lacoste PhD.)
- Exploit STARPU and PARSEC runtimes

Objectives
- Improve the scalability on heterogeneous platforms
- Express more parallelism thanks to the runtime → Change from 1D to 2D tasks
- PaStiX on top of STARPU with 2 Nvidia K40
- Limitation of the parallelism due to the 1D tasks approach
- Unbalanced workload between the GPUs and the CPUs
Heterogeneous architectures: from dense to sparse?  
From **Chameleon** to **PaStiX**?

**Example of audit**

- $N = 1000000$
- $\text{mean}(NB) = 200$
- $NT = 5000$
- 25 millions tiles $\rightarrow \approx 22\text{ billions tasks}$
- How many useful?
Heterogeneous architectures: from dense to sparse? 
From CHAMELEON to PaSTiX?

Example of audit

- \( N = 1000000 \)
- mean(\(NB\)) = 200
- \( NT = 5000 \)
- 25 millions tiles \(\rightarrow \approx 22\) billions tasks
- How many useful? \(\approx 1\) million
Trace on the audi with 2 K40

- The parallelism has been improved
- Better usage of the GPUs
- Less visible idle time than in the 1D version
- Might have some room for improvement with better scheduling strategy
Performance on the K40 architecture

![Graph showing performance on the K40 architecture]
Performance on the KNL architecture

![Performance graph on the KNL architecture](image-url)
Low-rank sparse direct solver

Context
- PASTIX sparse direct solver
- Collaboration with Univ. of Stanford
- G. Pichon Phd

Objectives
- Introduce block low-rank matrix format
- Reduce the computational complexity of the solver
- Reduce the memory complexity of the solver
Low-rank sparse direct solver: Two scenarios

Just-in-time

- Compress the block the latest as possible
- Do not reduce the memory peak
- Reduce the complexity and the time

MinimalMemory

- Compress as soon as possible
- Reduce the memory peak
- Kernels are more complex and less efficient
- Reduce the asymptotic complexity
Low-rank sparse direct solver: Memory consumption
Memory scalability on Laplacians

On miriel
- Problem of size $200^3 = 8M$ with full-rank
- Problem of size $330^3 = 36M$ with $10^{-4}$ tolerance
Memory scalability on Laplacians

On miriel

- Problem of size $200^3 = 8M$ with full-rank
- Problem of size $330^3 = 36M$ with $10^{-4}$ tolerance

On souris

- 96 cores, 3TB of memory
- $100 \times 100 \times 100K$ mesh
- The full-rank solver requires $11TB$
- With $\tau = 10^{-4}$:
  - Required Only $2TB$
  - Factorization time $< 6$ hours
  - 15 iterations to reach $10^{-8}$ solution
Conclusion

Many other softwares available on the gitlab (https://gitlab.inria.fr/solverstack):

- **MORSE_CMAKE**: Large set of cmake Find modules
- **HQR**: QR reduction trees
- **CHAMELEON**: Dense linear algebra subroutines for heterogeneous and distributed architectures
- **SPM**: Sparse Matrix package
- **PASTiX**: Parallel Sparse Direct Solver
- **FABULOUS**: Iterative Block Krylov Solvers
- **TABAKOV**: Task-BAsed KryLOV solver
- **PADDLE**: Parallel Algebraic Domain Decomposition for Linear systEms
- **H2P3S**: Hybrid High Performance Parallel Python Solver
- **MAPHY**: Massively Parallel Hybrid Solver
- **HIPS**: Hierarchical Iterative Parallel Solver
- **SCALFMM**: N-body simulation using kernel independent Fast Multipole Method
- **ViTE**: Visualization Trace Explorer

Thank you !!!