MORSE

Matrices Over Runtime Systems @ Exascale

Emmanuel Agullo, Olivier Aumage, George Bosilca, Bérenger Bramas, Alfredo Buttari
Olivier Coulaud, Éric Darve, Jack Dongarra, Mathieu Faverge, Nathalie Furmento
Luc Giraud, Abdou Guermouche, Julien Langou, Florent Lopez, Hatem Ltaief
Samuel Pitoiset, Florent Pruvost, Marc Sergent, Samuel Thibault, Stanimire Tomov

The MORSE project

The goal of the Matrices Over Runtime Systems @ Exascale (MORSE) project is to design dense and sparse linear algebra methods that achieve the fastest possible time to an accurate solution on large-scale multicore systems with GPU accelerators, using all the processing power that future high end systems can make available. We propose a framework for describing matrix algorithms with a sequential expression at a high level of abstraction and delegating the actual execution to a runtime system. In this poster we show that this model allows for (1) achieving an excellent scalability on heterogeneous clusters, (2) designing advanced numerical algorithms and (3) being compliant with standards such as OpenMP 4.0 or possible extensions of this standard. We illustrate our methodology on three classes of problems: dense linear algebra, sparse direct methods and fast multipole methods. The resulting codes have been incorporated into the Chameleon, QR_MUMPS and ScalFMM solvers, respectively.

Associate teams

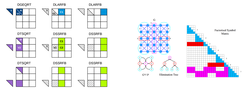
FastLA Inria Associate Team
Inria HiePACS team
Lawrence Berkeley National Lab
Stanford University
http://www.inria.fr/en/teams/fastla

MORSE Inria Associate Team
Inria HiePACS and Storm teams
KAUST Supercomputing Laboratory
The University of Colorado
The University of Tennessee, Knoxville
http://www.inria.fr/en/teams/morse

Design principles

Linear Algebra

$Ax = b$



Dense

Sparse

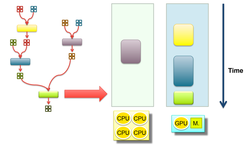
Sequential-Task-Flow

```
FOR k = 0..TILES-1
  A[k][k], T[k][k] ← DGEQRT(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])
```

Direct Acyclic Graph

- Fine granularity
- Asynchronous
- DAG to schedule

Runtime systems



- Schedule tasks onto PUs
- Ensure data availability
- Ensure data coherency
- Offload computation

▷ **Exploit all ressources**
▷ **Hide low-level complexity**
▷ **Performance portability**

Softwares: StarPU, Kaapi, Quark, PaRSEC, StarSS

Optimized kernels

CPUs	GPUs
BLAS, LAPACK	CUDA/cuBLAS, MAGMA

laptop

accelerator

cluster

▷ **Address a wide range of computing platforms**

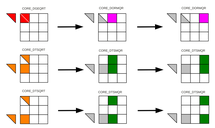
Chameleon: Matrix algebra for heterogeneous architectures

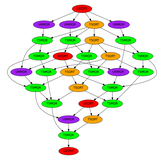
Download the library at `https://project.inria.fr/chameleon/`

Chameleon is a dense linear algebra software similar to LAPACK but for heterogeneous/hybrid distributed architectures. We present here how Chameleon has been extended using the StarPU / Quark / PaRSEC runtime systems in order to handle multicore nodes enhanced with multiple GPUs and clusters of interconnected nodes. Chameleon includes one-sided factorizations (LLt, LU, QR) and relative solvers as well as main level-3 BLAS operations.

TILE ALGORITHMS

Similarly to PLASMA, the matrix is split in square blocks called tiles. Based on such a layout, tile algorithms aim at improving data locality and increase parallelism. The operations performed on those tiles are executed using optimized CPU and GPU kernels. When GPU are turned on, the tile size is increased to fully benefit from the potential of the accelerators.

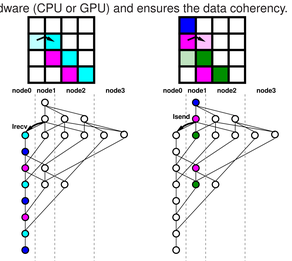




The high-level algorithm (here a QR factorization) can then be represented as a task graph where the nodes represent the tasks in which it is decomposed and the edges represent the dependencies among them. The runtime system (here StarPU) dynamically schedules the tasks on the available hardware (CPU or GPU) and ensures the data coherency.

MPI DISTRIBUTED VERSION

- All nodes unroll the whole task graph
- They determine tasks they will execute
- They can infer required communications
- No negociation between nodes (not master-slave)
- Unrolling can be pruned



Node 0 execution | Node 1 execution

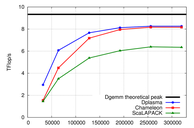
CHAMELEON-1.0

- Multicore + multi-GPU (Cuda)
- Distributed Memory Machines
- Multiple Precision Support
- Level-3 BLAS
- Solution of Linear Equations
- Linear Least squares
- User's guide and tutorial

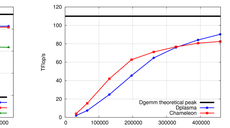
NEXT STEPS

- Eigenvalue Problems
- Singular Value Decomposition
- OpenCL
- Intel MIC Accelerators

PERFORMANCE RESULTS (Dense Cholesky Factorization)



144 nodes of two quad-core Intel Xeon X5620 (8 CPUs)



144 nodes of two quad-core Intel Xeon X5620 + two Nvidia Tesla M2090 GPU

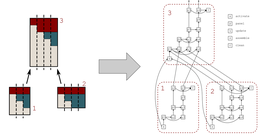
QR_MUMPS: Multifrontal QR for GPU and multicore architecture

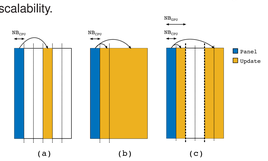
More information at `http://buttari.perso.enseeiht.fr/qr_mumps/`

The qr_mumps QR solver is a software package for the solution of sparse, linear systems on multicore computers. It implements a direct solution method based on the QR factorization of the input matrix. It has been extended with the support of the StarPU runtime system for handling multicore platforms enhanced with a GPU.

MULTIFRONTAL TILE ALGORITHM

Parallelism is achieved through a decomposition of the workload into fine-grained computational tasks which basically correspond to the execution of a BLAS or LAPACK operation on a block-column. Parallelism is achieved by dividing the workload into fine grained tasks that are arranged in a Direct Acyclic Graph (DAG). The model allows for pipelining tasks at different levels in the tree, which provides high efficiency and scalability.





In the multicore case, a fine grain decomposition is applied. Picture (a) on the left shows for instance a 1D fine grain decomposition. To better benefit from accelerators, a coarser grain may be used (b) such as in the case of the dense linear algebra MAGMA reference library. Finally, to better cope with heterogeneous platforms, a hierarchical decomposition is employed (c).

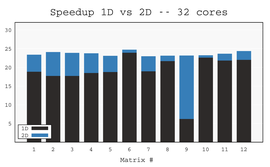
QR_MUMPS

- Multicore + 1 GPU (Cuda)
- Multiple Precision Support
- Memory-Aware algorithm
- 1D/2D Multicore Tile Algorithms
- Hierarchical Heterogeneous Partitioning

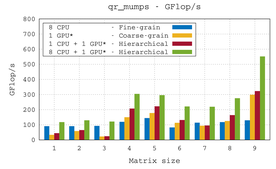
NEXT STEPS

- Multi-GPU Support
- Distributed Memory Machines
- PaRSEC Support
- SimGrid Support

PERFORMANCE RESULTS



Four octo-core Intel Sandy Bridge E5-4650 (32 CPUs)



One octo-core Intel Xeon E5-2650 (8 CPUs) + One Nvidia Kepler K40c GPU

qr_mumps is a project of the APO team of the IRIT laboratory, co-developed with Inria-HiePACS

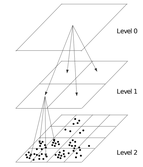
ScalFMM: Parallel fast multipole library for large scale simulations

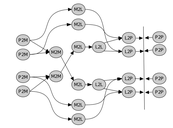
Download the library at `http://scalfmm.gforge.inria.fr/`

The Parallel Fast Multipole Library for Large Scale Simulations (ScalFMM) library aims at simulating N-body interactions or matrix-vector products using the Fast Multipole Method (FMM). This software intends to offer all the functionalities needed to perform large parallel simulations while enabling an easy customization of the simulation components: kernels, particles and cells. The current MORSE distribution of ScalFMM proposes FMM based on interpolation.

INTERPOLATION BASED FMM

Our FMM approach is kernel-independent based on a Chebyshev or a Lagrange interpolation. We have designed the corresponding FMM operators and implemented them on CPU: P2M (particle-to-moment), M2M (moment- to-moment), L2L (local-to-local), L2P (local-to-particle), M2L (moment-to-local) and P2P (particle-to-particle).

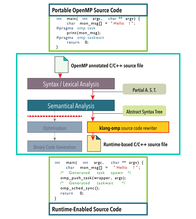




The high-level algorithm can then be represented as a task graph where the nodes represent these operators. The P2P and M2L operators dominate the computational cost of the whole algorithm. We have also designed optimized versions of these operators for GPUs. The runtime system can then dynamically schedule them on a CPU or GPU to balance the load and maximize the performance.

Klang-OMP OpenMP Compiler

- Source-to-source OpenMP C/C++ Compiler
- Supports OpenMP 3.1, OpenMP 4.0 dependent tasks
- Generates runtime system calls from directives
- Enables runtime abstraction, preserves runtime performances
- Supports runtime-specific extensions (priorities, commute, . . .)



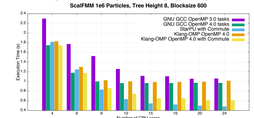
SCALFMM-1.4

- Multicore + multi-GPU (Cuda)
- Distributed Memory Machines
- Multiple Precision Support
- Chebyschev and uniform grid points FMM
- Kernel Independent
- User's guide and examples

NEXT STEPS

- Adaptative algorithm
- Anisotropic Kernels for Molecular Dislocation
- Oscillatory Kernels
- OpenCL
- Intel MIC Accelerators

PERFORMANCE RESULTS

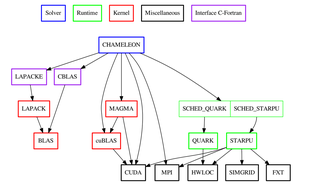


Two dodeca-core Intel Haswell Xeon E5-2680 (24 CPUs)

MORSE Distribution

The innovative methodology employed to design solvers over runtime systems increases the number of dependencies in the software stack, which may be prohibitive for the software diffusion.

The distribution tool, *Spack* a package management tool convenient for HPC libraries, handles this complexity by ensuring the deployment and the coherency of all the components on UNIX systems.



Example of dependencies tree: **Chameleon** dense solver

MORSE IN *SPACK*

Download at `https://github.com/fpruvost/spack/` - **morse branch**

- Language = Python: no third-party tool to install, ready to be used

```
$ git clone git@github.com:fpruvost/spack.git && ./spack/bin/spack install mpi
```

- Easy way to give build variants

```
$ spack install chameleon%gcc4.8.4+starpu+mpi+cuda ^starpu@1.2+mpi+cuda
```

- Handle modulefiles and local filesystem mirrors

```
$ spack mirror create openmpi hwloc fxt starpu
```

MORSE FEATURES

- Cluster of multicore + multi-GPU Nodes
- Dense and Sparse Linear Algebra
- Fast Multipole Method
- Multiple Precision support
- User's Guide and Tutorial
- Installation Support on UNIX System:

solvers	chameleon, pastix, qr_mumps, scalfmm, ...
runtimes	quark, parsec, starpu
kernels	(c)blas, lapack(e), fftw
misc	(par)metis, (pt)scotch, hwloc, fxt, simgrid

NEXT STEPS

- Two-sided Dense Solvers
- H-Matrix Solvers
- Sparse Hybrid Direct / Iterative Solvers
- OpenCL, Intel MIC Accelerators
- Advanced Scheduling Algorithms
- Detection Mode In *Spack*
- Integrate Packages in *Spack* releases