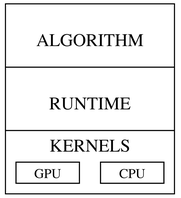
The **MORSE** Project

MATRICES OVER RUNTIME SYSTEMS @ EXASCALE

Emmanuel Agullo, Éric Darve, Xavier Lacoste, Pierre Ramet
George Bosilca, Jack Dongarra, Julien Langou, Toru Takahashi
Bérenger Bramas, Mathieu Faverge, Hatem Ltaief, Samuel Thibault
Cédric Castagnède, Nathalie Furmento, Matthias Messner, Stanimire Tomov
Olivier Coulaud, Luc Giraud, Raymond Namyst, Ichitaro Yamazaki

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 at a high level of abstraction and delegating the actual execution to a runtime system in order to design software whose performance is portable across architectures. 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 MAGMA, PaStiX and ScalFMM solvers, respectively.



FastLA Inria Associate Team

Inria HiePACS team
Lawrence Berkeley National Lab
Stanford University

MORSE Inria Associate Team

Inria HiePACS and RUNTIME teams
KAUST Supercomputing Laboratory
The University of Colorado
The University of Tennessee

MATRIX ALGEBRA ON GPU AND MULTICORE ARCHITECTURES

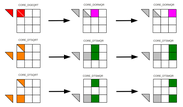
MAGMA

DOWNLOAD THE LIBRARY AT **http://icl.utk.edu/magma/**

The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures. We present here how MAGMA has been extended using the StarPU / Quark runtime systems in order to handle multicore nodes enhanced with multiple GPUs. The MORSE extension of MAGMA 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 into square blocks called tiles. Based on such a layout, tile algorithms aim at improving data locality and increasing parallelism. The operations performed on those tiles are executed using optimized CPU and GPU kernels. When the GPUs 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 dependences among them. The runtime system (here StarPU) dynamically schedules the tasks on the available hardware (CPU or GPU) and ensures the data coherency.

MAGMA-MORSE 1.3

- Multicore + multi-GPU (CUDA)
- Multiple Precision Support
- Level-3 BLAS
- Solution of Linear Equations
- Linear Least Squares (CPU only)
- LAPACK and Native Interfaces
- LAPACK-Derived Testing Suite
- MAGMA-MORSE Users' Guide

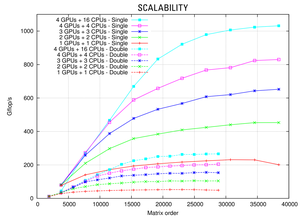
NEXT STEPS

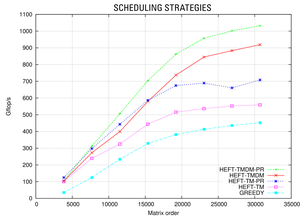
- Linear Least Squares (GPUs)
- Singular Value Decomposition
- Symmetric and Non Symmetric Eigenvalue Problems
- Communication Avoiding
- OpenCL, Intel MIC Accelerators
- MPI Interface

PERFORMANCE RESULTS

DENSE QR FACTORIZATION

CPU Quad-socket quad-core AMD Opteron SE 8358 (16 CPU)
GPU NVidia S1070 Tesla accelerators (4 GPU)





PARALLEL SPARSE MATRIX PACKAGE

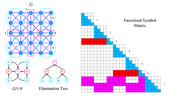
PaSTIX

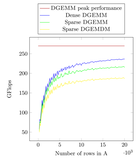
DOWNLOAD THE LIBRARY AT **http://pastix.gforge.inria.fr/**

The Parallel Sparse matriX package (PaStiX) is a scientific library that provides a high performance parallel solver for very large sparse linear systems based on direct methods. Numerical algorithms are implemented in single or double precision (real or complex) using LLt, LDLt and LU factorizations with static pivoting. We also provide an adaptive blockwise iLU(k) factorization. Except usage of distributed memory machines, most PaStiX functionalities are available in the MORSE distribution.

SUPERNODAL METHOD

Sparse direct methods are a variant of Gaussian elimination for sparse linear systems. A nonzero variable can be represented as an edge within the graph (G) associated with the matrix. In supernodal methods, this graph is partitioned into sets of variables (supernodes) that can be eliminated simultaneously .





The supernodal factorization consists of two types of tasks: factorization of the current supernode and supernode-supernode updates. The latter one is a 200 variant of a matrix-matrix multiplication dealing with indirections. Its GPU implementation is crucial for the 100 performance of the whole factorization. To 50 efficiently handle indirections, a new GPU Sparse DGEMM has been designed.

PaSTIX-MORSE 5.3

- Multicore + multi-GPU (CUDA)
- Multiple Precision Support
- LLt, LDLt, LU factorizations
- Static Pivoting
- Multiple RHS
- External orderings (PT-Scotch / METIS)
- PaStiX-MORSE Users' Guide

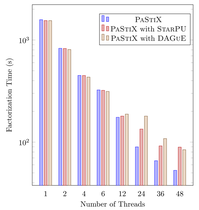
NEXT STEPS

- Schur Complement Computation
- Krylov Iterative Refinement
- Granularity Optimization
- Dynamic Splitting of the Tasks
- Static Hints for Scheduling
- OpenCL, Intel MIC Accelerators
- Distributed Memory Machines

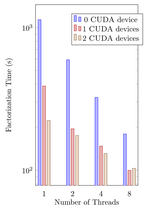
PERFORMANCE RESULTS

SPARSE LU FACTORIZATION (MHD Matrix - 486K unknowns)

CPU Quad-socket dodeca-core AMD Opteron SE 6180 (48 CPU)



CPU Dual-socket hexa-core Intel X5650 (12 CPU)
GPU NVIDIA M2090 Fermi accelerators (0 to 2 GPU)



PARALLEL FAST MULTIPOLE LIBRARY FOR LARGE SCALE SIMULATIONS

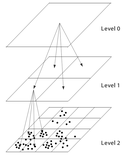
ScalFMM

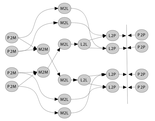
DOWNLOAD THE LIBRARY AT **http://scalfmm.gforge.inria.fr/**

The Parallel Fast Multipole Library for Large Scale Simulations (ScalFMM) library aims at simulating N-body interactions 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 a Chebyshev interpolation.

CHEBYSHEV FMM

Our FMM approach is kernel-independent based on a Chebyshev 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.

ScalFMM-MORSE 1.0

- Multicore + multi-GPU (CUDA)
- Multiple Precision Support
- Chebyschev FMM
- P2P and M2L Optimized GPU kernels
- Kernel Independent
- ScalFMM-MORSE Users' Guide

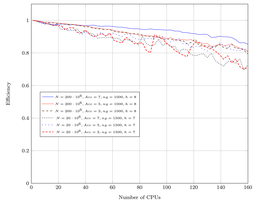
NEXT STEPS

- Spherical and Cartesian Expansions
- Isotropic and Anisotropic Kernels for Molecular Dislocation
- Oscillatory Kernels
- OpenCL, Intel MIC Accelerators
- Distributed Memory Machines

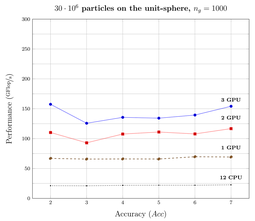
PERFORMANCE RESULTS

CHEBYSHEV FMM

CPU Twenty octa-core Intel Xeon E7-8837 (160 CPUs)



CPU Dual-socket hexa-core Intel X5650 (12 CPU)
GPU NVIDIA M2090 Fermi accelerators (0 to 2 GPU)



The **MORSE** Software Distribution

DOWNLOAD THE LIBRARY AT **http://www.inria.fr/en/teams/morse/**

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 Morse_Distrib tool handles this complexity by ensuring the deployment and the coherency of all the components.

MORSE 1.0

- Multicore + multi-GPU (CUDA)
- Multiple Precision Support
- Level-3 BLAS
- One-sided Dense Solvers
- Supernodal Sparse Direct Method
- Chebyshev Fast Multipole Method
- Unified Runtime System (StarPU)
- Support of hardware locality (hwloc)
- Partial Support of Quark and DAGuE Runtime Systems
- Unified Software Distribution
- MORSE Users' Guide

NEXT STEPS

- Two-sided Dense Solvers
- Stencil Computation
- Multifrontal Sparse Direct Methods
- H-Matrix Solvers
- Sparse Hybrid Direct / Iterative Solvers
- OpenCL, Intel MIC Accelerators
- Distributed Memory Machines
- Advanced Scheduling Algorithms