DPLASMA (Distributed Parallel Linear Algebra Software for Multicore Architectures) is the leading implementation of a dense linear algebra package for distributed heterogeneous systems. It is designed to deliver sustained performance for distributed systems where each node features multiple sockets of multicore processors, and if available, accelerators like GPUs or Intel Xeon Phi. DPLASMA achieves this objective through the state of the art PaRSEC runtime, porting the Parallel Linear Algebra Software for Multicore Architectures (PLASMA) algorithms to the distributed memory realm.

**FUNCTIONALITY**  

<table>
<thead>
<tr>
<th>Linear Systems of Equations</th>
<th>Cholesky, LU (inc. pivoting, PP), LDL (prototype)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Least Squares</td>
<td>QR &amp; LQ</td>
</tr>
<tr>
<td>Symmetric Eigenvalue Problem</td>
<td>Reduction to Band (prototype)</td>
</tr>
<tr>
<td>Level 3 Tile BLAS</td>
<td>GEMM, TRSM, TRMM, HEMM/SYMM, HERK/SYRK, HER2/SYR2K</td>
</tr>
</tbody>
</table>

**FEATURES**  

- Covering four precisions: double real, double complex, single real, single complex (D, Z, S, C)
- Providing ScalAPACK-compatible interface for matrices in F77 column-major layout
- Supporting: Linux, Windows, Mac OS X, UN*X (depends on MPI, hwloc)
- Fine-grain Composition of Operations
- Two-sided Factorizations
- Distributed Sparse Solver
- More GPU kernels integration
- LU+RBT

**USER DEFINED DATA PLACEMENT**

In addition to traditional ScalAPACK data distribution, DPLASMA provides interfaces for users to expose arbitrary tile distributions, and the algorithms transparently operate on local data, or introduce implicit communications to resolve dependencies, removing the burden of initial data re-shuffle, and providing to the user a novel approach to address load balance.

**FUTURE PLANS**

- Fine-grain Composition of Operations
- Two-sided Factorizations
- Distributed Sparse Solver
- More GPU kernels integration
- LU+RBT

**FUNCTIONALITY COVERAGE**

- **Linear Systems of Equations**
  - Cholesky, LU (inc. pivoting, PP), LDL (prototype)
- **Least Squares**
  - QR & LQ
- **Symmetric Eigenvalue Problem**
  - Reduction to Band (prototype)
- **Level 3 Tile BLAS**
  - GEMM, TRSM, TRMM, HEMM/SYMM, HERK/SYRK, HER2/SYR2K

**ENERGY EFFICIENCY**

Solving Linear Least Square Problem (DGEQRF)
System G Virginia Tech, 32-node, 256-core, Intel Xeon 2.8GHz, IB20G
PaRSEC DPOTRF (on 64 nodes, 16 cores+3GPUs/node)
PaRSEC DPOTRF (on 64 nodes, 16 cores+2GPUs/node)
PaRSEC DPOTRF (on 64 nodes, 16 cores+1GPU/node)

**DISTRIBUTED HYBRID DPOTRF**

Problem Scaling on Keeneland
64 nodes (1024 cores, 192 M2090 GPUs, Infiniband 20G)
PaRSEC DPOTRF on 1 node: 1187Gflop/s
MAGMA DPOTRF on 1 node: 1155Gflop/s

**IN COLLABORATION WITH**

INSTITUTE FOR INNOVATIVE COMPUTING LABORATORY  
INSTITUTE OF INFORMATION TECHNOLOGY RESEARCH
University of Tennessee

**WITH SUPPORT FROM**

Microsoft
KAUST
National Science Foundation

**SPONSORED BY**

National Science Foundation

FIND OUT MORE AT http://icl.cs.utk.edu/dplasma
PaRSEC is a generic framework for architecture aware scheduling and management of micro-tasks on distributed many-core heterogeneous architectures. Applications we consider can be expressed as a Direct Acyclic Graph of tasks with labeled edges designating data dependencies. DAGs are represented in a compact problem-size independent format that can be queried on-demand to discover data dependencies in a totally distributed fashion. PaRSEC assigns computation threads to the cores, overlaps communications and computations and uses a dynamic, fully-distributed scheduler based on architectural features such as NUMA nodes and algorithmic features such as data reuse.

**PaRSEC TOOLCHAIN**

- **Programmer**
- **Serial Code**
- **Dataflow representation**
- **Domain Specific Extensions**
- **Application code & Codelets**
- **Runtime**
- **Supercomputer**
- **Additional libraries**
- **System compiler**
- **MPA, pthreads, CUDA, PLASMA, MAGMA**
- **PaRSEC compiler**
- **PaRSEC Toolchain**
- **Parallel tasks stubs**
- **Data distribution**

**Features**

- Supports Distributed Heterogeneous Platforms
- Sustained Performance
- NUMA & Cache Aware Scheduling
- State-of-the-art Algorithms
- Capacity Level Scalability
- Performance Portability
- Implicit Communication
- Communication Overlapping

**Input serial codes are converted automatically by the PaRSEC compiler into the task Dataflow representation which can also be edited by the programmer. The Dataflow compiler generates the stubs that, along with the Data distribution provided by the programmer via Domain Specific Extensions, the Application code & Codelets, the Runtime and relevant libraries are linked by the system compiler to generate the executable that will run on a heterogeneous distributed memory supercomputer.**

**PaRSEC uses a symbolic, problem size independent representation to express the Directed Acyclic Graph (DAG) of the Dataflow of a program. As a result, at runtime, successors and predecessors of any given task can be evaluated independently, without exploring portions of the DAG pertaining to tasks localized on other nodes. Furthermore, the whole DAG is never unfolded, and only the set of locally active tasks resides in the memory at any given time.**

**AUTOTUNING (MULTI-LEVEL)**

- **1 Nodes (8 cores)**
- **4 Nodes (32 cores)**
- **81 Nodes (648 cores)**

![Block Size (NB) vs % efficiency graph](http://icl.utk.edu/parsec)