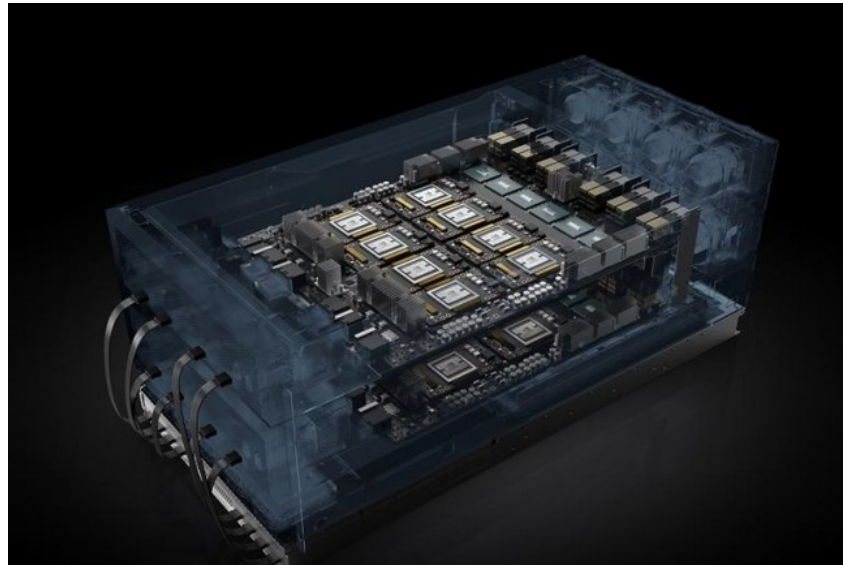
RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

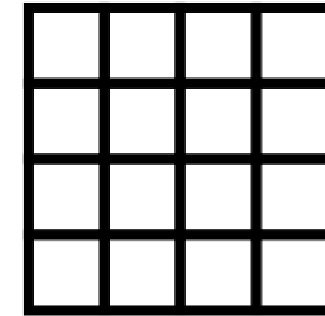
Inria

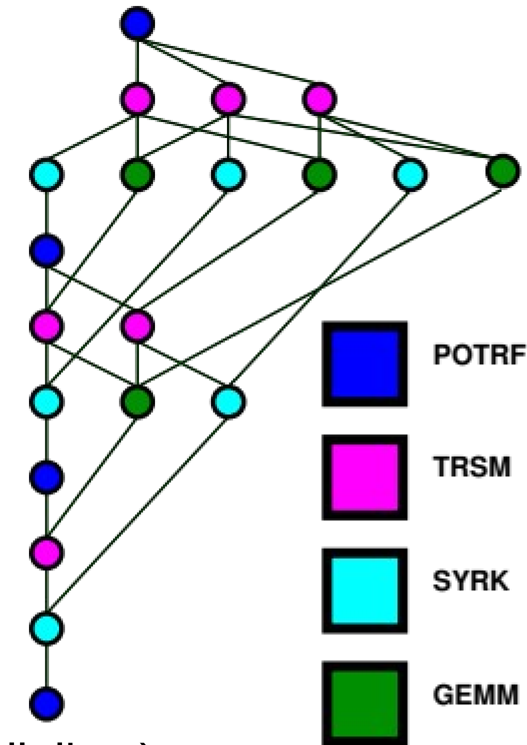
LINEAR ALGREBRA ALGORITHMS ON TOP OF STARPU RUNTIME SYSTEM



Chameleon: dense linear algebra library

- Written in C, Fortran interface, CMake
- Algorithms: TR/GEADD, SY/HE/GEMM, PO/GETRF, GEQR/LQF, GELS, …
- Matrices forms: general, symmetric, triangular
- Precisions: s, d, c, z
- Runtime systems: OpenMP, PaRSEC, StarPU
- Distributed MPI
- CUDA /HIP
- In development: compression, mixed precision, python interface

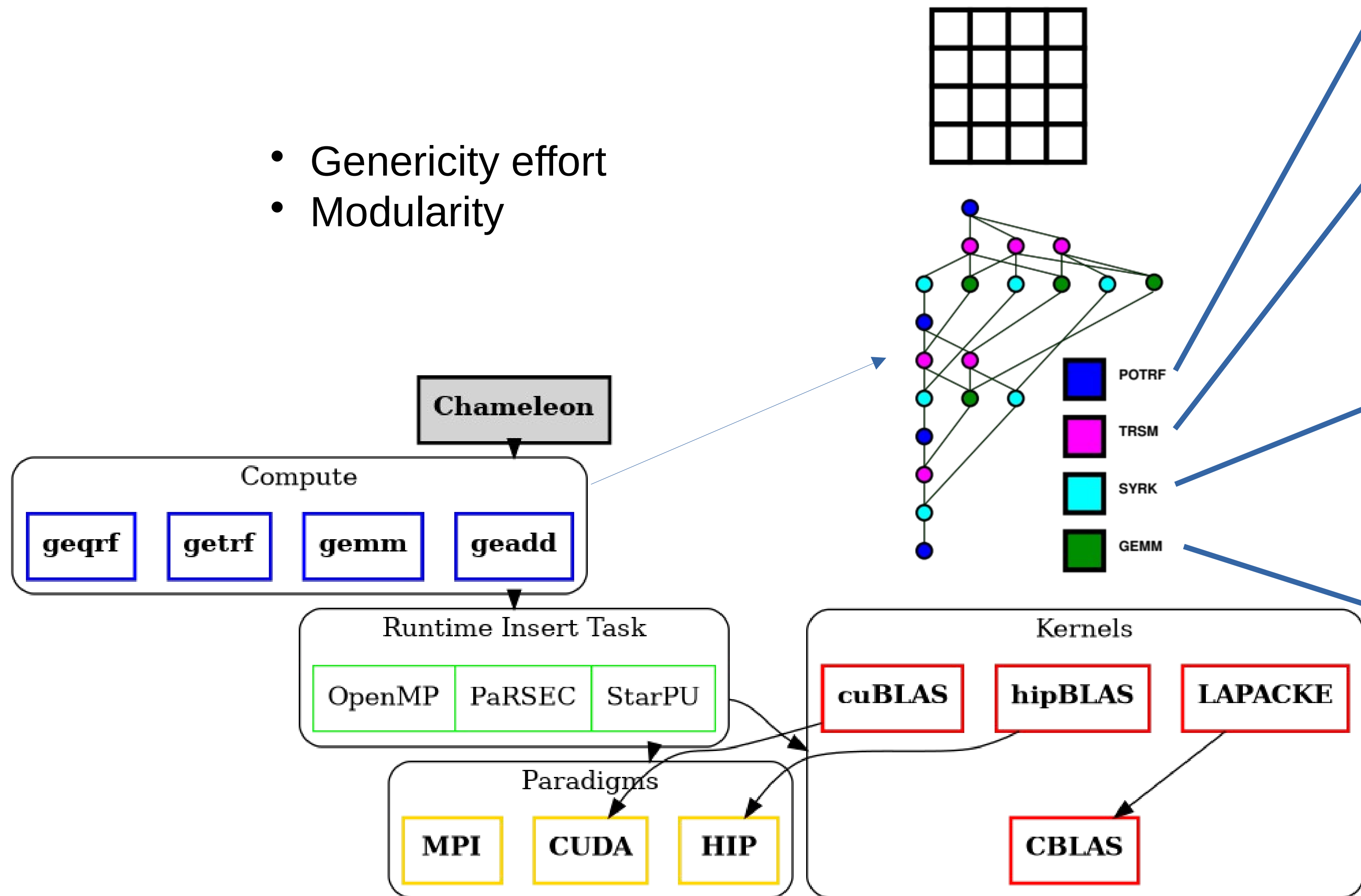




Philosophy:
- Write algorithms generically, not paradigm or vendor specific (cuda, mkl, ...)
- Choose size and kernels on sub-blocks to maximize Gflops rate (coarse grain parallelism)
- Regular block size, square (m=n) and fixed → ease algorithms development

Chameleon dense linear library
Code structure : algorithms

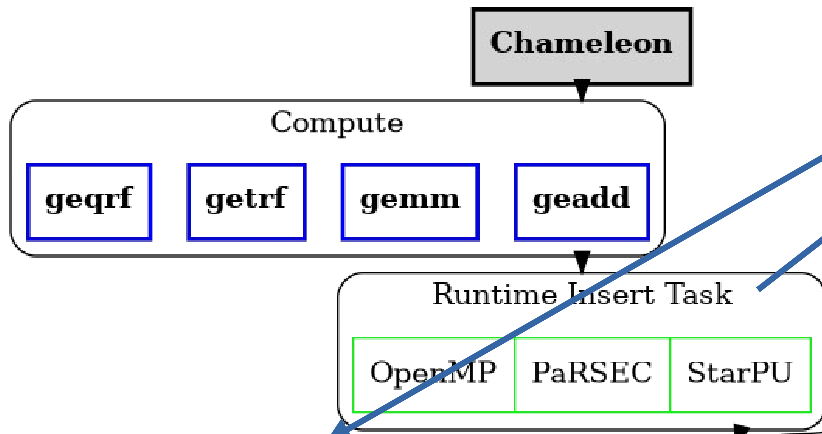
- Genericity effort
- Modularity




```
for (k = 0; k < A->mt; k++) {
    tempkm = k == A->mt-1 ? A->m-k*A->mb : A->mb;
    INSERT_TASK_zpotrf(
        &options,
        ChamLower, tempkm, A->mb,
        A(k, k), A->nb*k);
    for (m = k+1; m < A->mt; m++) {
        tempmm = m == A->mt-1 ? A->m-m*A->mb : A->mb;
        INSERT_TASK_ztrsm(
            &options,
            ChamRight, ChamLower, ChamConjTrans, ChamNonUnit,
            tempmm, A->mb, A->mb,
            zone, A(k, k),
                  A(m, k));
    }
    for (n = k+1; n < A->nt; n++) {
        tempnn = n == A->nt-1 ? A->n-n*A->nb : A->nb;
        INSERT_TASK_zherk(
            &options,
            ChamLower, ChamNoTrans,
            tempnn, A->nb, A->mb,
            -1.0, A(n, k),
             1.0, A(n, n));
        for (m = n+1; m < A->mt; m++) {
            tempmm = m == A->mt-1 ? A->m - m*A->mb : A->mb;
            INSERT_TASK_zgemm(
                &options,
                ChamNoTrans, ChamConjTrans,
                tempmm, tempnn, A->mb, A->mb,
                mzone, A(m, k),
                       A(n, k),
                zone,  A(m, n));
        }
    }
}
```

Chameleon dense linear library
Code structure : insert tasks

```c
    /* Insert the task */
    rt_starpu_insert_task(
        &cl_zgemm,
        /* Task codelet arguments */
        STARPU_CL_ARGS, clargs, sizeof(struct cl_zgemm_args_s),

        /* Task handles */
        STARPU_R, RTBLKADDR(A, ChamComplexDouble, Am, An),
        STARPU_R, RTBLKADDR(B, ChamComplexDouble, Bm, Bn),
        accessC,  RTBLKADDR(C, ChamComplexDouble, Cm, Cn),

        /* Common task arguments */
        STARPU_PRIORITY,          options->priority,
        STARPU_CALLBACK,          callback,
        STARPU_EXECUTE_ON_WORKER, options->workerid,
        STARPU_POSSIBLY_PARALLEL, options->parallel,
#if defined(CHAMELEON_CODELETS_HAVE_NAME)
        STARPU_NAME,              cl_name,
#endif
        0 );
```



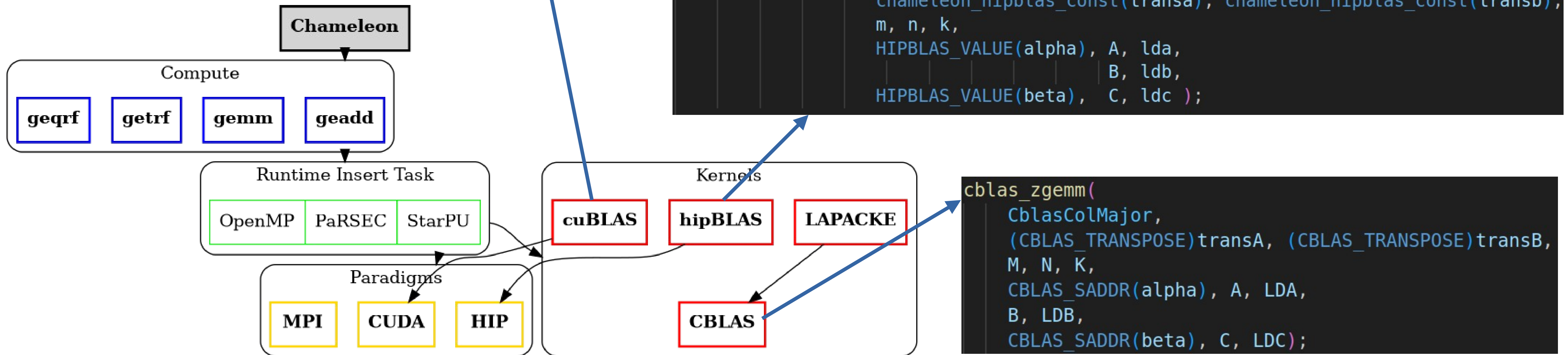

```c
static void
cl_zgemm_cpu_func( void *descr[], void *cl_arg )
{
    struct cl_zgemm_args_s *clargs = (struct cl_zgemm_args_s *)cl_arg;
    CHAM_tile_t *tileA;
    CHAM_tile_t *tileB;
    CHAM_tile_t *tileC;

    tileA = cti_interface_get(descr[0]);
    tileB = cti_interface_get(descr[1]);
    tileC = cti_interface_get(descr[2]);

    TCORE_zgemm( clargs->transA, clargs->transB,
                 clargs->m, clargs->n, clargs->k,
                 clargs->alpha, tileA, tileB,
                 clargs->beta,  tileC );
}
```

```c
static void
cl_zgemm_cuda_func( void *descr[], void *cl_arg )
{
    struct cl_zgemm_args_s *clargs = (struct cl_zgemm_args_s *)cl_arg;
    cublasHandle_t          handle = starpu_cublas_get_local_handle();
    CHAM_tile_t *tileA;
    CHAM_tile_t *tileB;
    CHAM_tile_t *tileC;

    tileA = cti_interface_get(descr[0]);
    tileB = cti_interface_get(descr[1]);
    tileC = cti_interface_get(descr[2]);

    assert( tileA->format & CHAMELEON_TILE_FULLRANK );
    assert( tileB->format & CHAMELEON_TILE_FULLRANK );
    assert( tileC->format & CHAMELEON_TILE_FULLRANK );

    CUDA_zgemm(
        clargs->transA, clargs->transB,
        clargs->m, clargs->n, clargs->k,
        (cuDoubleComplex*)&(clargs->alpha),
        tileA->mat, tileA->ld,
        tileB->mat, tileB->ld,
        (cuDoubleComplex*)&(clargs->beta),
        tileC->mat, tileC->ld,
        handle );
}
```

```c
static void
cl_zgemm_hip_func( void *descr[], void *cl_arg )
{
    struct cl_zgemm_args_s *clargs = (struct cl_zgemm_args_s *)cl_arg;
    hipblasHandle_t         handle = starpu_hipblas_get_local_handle();
    CHAM_tile_t *tileA;
    CHAM_tile_t *tileB;
    CHAM_tile_t *tileC;

    tileA = cti_interface_get(descr[0]);
    tileB = cti_interface_get(descr[1]);
    tileC = cti_interface_get(descr[2]);

    assert( tileA->format & CHAMELEON_TILE_FULLRANK );
    assert( tileB->format & CHAMELEON_TILE_FULLRANK );
    assert( tileC->format & CHAMELEON_TILE_FULLRANK );

    HIP_zgemm(
        clargs->transA, clargs->transB,
        clargs->m, clargs->n, clargs->k,
        (hipblasDoubleComplex*)&(clargs->alpha),
        tileA->mat, tileA->ld,
        tileB->mat, tileB->ld,
        (hipblasDoubleComplex*)&(clargs->beta),
        tileC->mat, tileC->ld,
        handle );

    return;
}
```

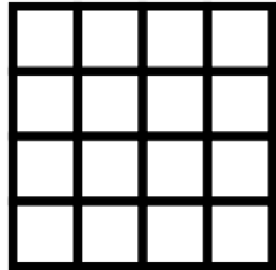
Chameleon dense linear library
Code structure : kernels

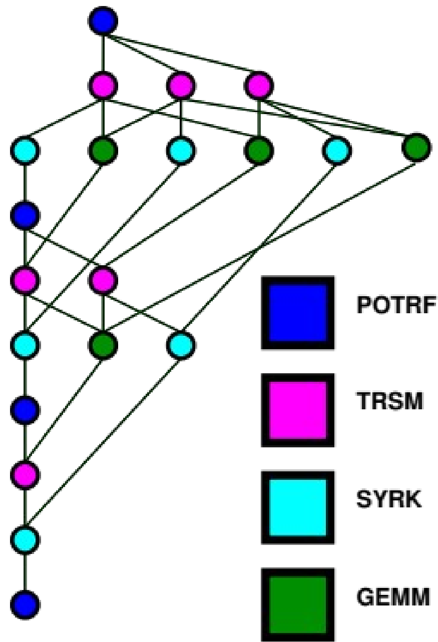
```
rc = cublasZgemm( handle,
                  chameleon_cublas_const(transa), chameleon_cublas_const(transb),
                  m, n, k,
                  CUBLAS_VALUE(alpha), A, lda,
                                       B, ldb,
                  CUBLAS_VALUE(beta),  C, ldc);
```



Key performance issues

Workers computing speed heterogeneity : CPU vs. GPU





On modern machines:

- block size for CPUs: ~500
- block size for GPUs: ~2000
- many CPUs: ~64
- a few GPUs: ~4

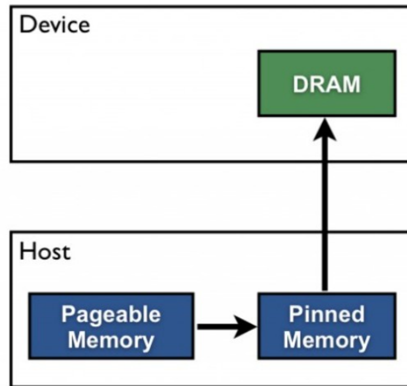
To handle this complexity StarPU propose two features:

- Parallel tasks (MKL)
  - Use GPU block size (~2000)
  - Execute CPUs kernels (multithreaded) on clusters of several cores (e.g. 8)
- Recursive tasks (in development)

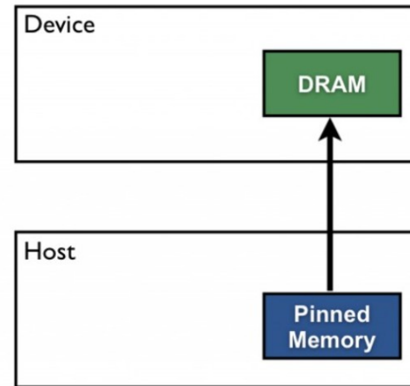
Key performance issues
Data transfers considerations

Memory pinning for faster host – device data transfers (x2)

Pageable Data Transfer

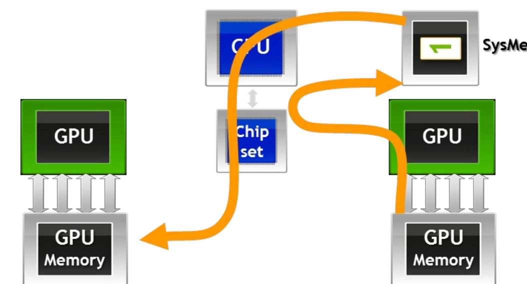


Pinned Data Transfer

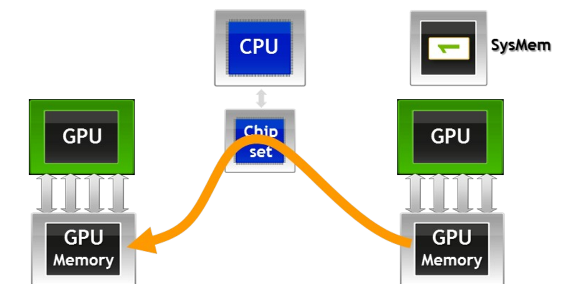


GPUDirect for faster device – device data transfers (x10)

No GPUDirect P2P



GPUDirect P2P



StarPU handles this automatically, but:

- pinning causes extra allocation cost when initializing large data (matrices)
  - avoid if no data reuse
- be sure to have GPUDirect available, also with MPI
  - run benchmarks with simple StarPU examples

Interesting features

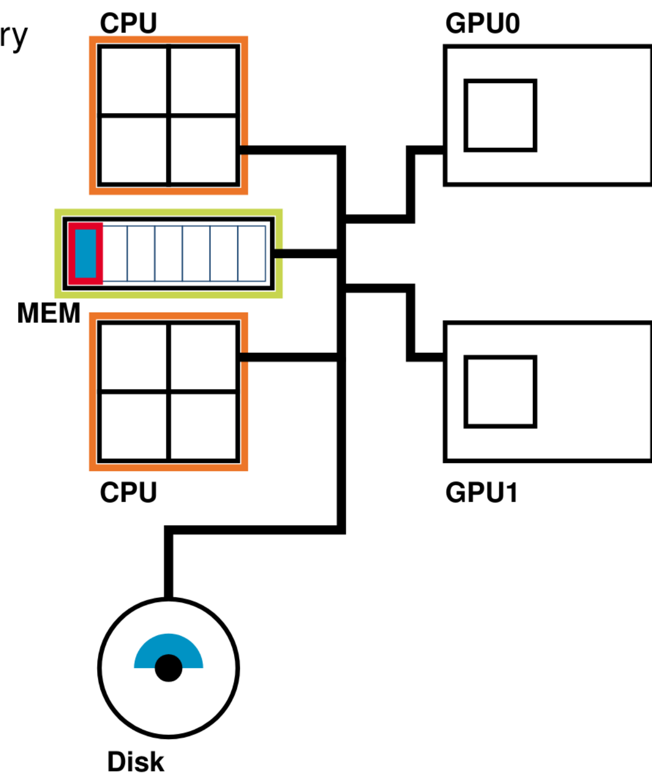
Out-of-Core

Integration with general StarPU's memory management layer

- StarPU data handles
- Task dependencies
  - Data reloaded automatically

Multiple disk drivers supported

- Legacy stdio/unistd methods
- Google's LevelDB (key/value database library)



Simulation with SimGrid

Scheduling without executing kernels

- Requires the SimGrid simulation environment
- Enables simulating large-scale scenarios
  - Large data sets
  - Large simulated hardware plaform
- Relies on **real** performance models...
- ...collected by StarPU on a real machine
- Enables fast experiments when designing application algorithms
- Enables fast experiments when designing scheduling algorithms

```
1 $ $STARPU_DIR/configure --enable-simgrid [... other opts ...]
2 ...
```

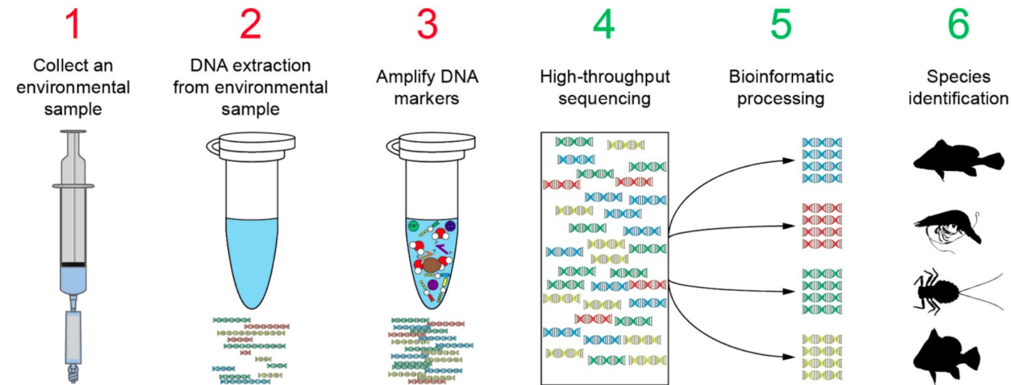
Interesting features

Execution traces

Chameleon GEMM followed by QR on 8 CPUs and 2 GPUs

Application example : Multidimensional Scaling (MDS)

Metabarcoding



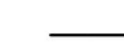
Multidimensional Scaling (think PCA)

Matrice de distance D

$$\begin{pmatrix} d_{1,1} & d_{1,2} & \cdots & d_{1,n} \\ d_{2,1} & d_{2,2} & \cdots & d_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ d_{n,1} & d_{n,2} & \cdots & d_{n,n} \end{pmatrix}$$

Projection sur un espace Euclidien

$r \ll n$



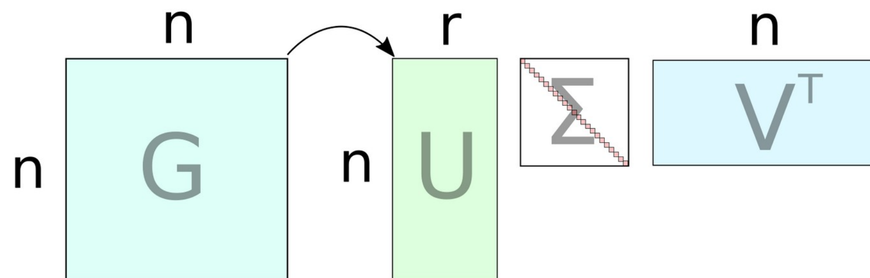
$\longrightarrow$

Conservation des distances

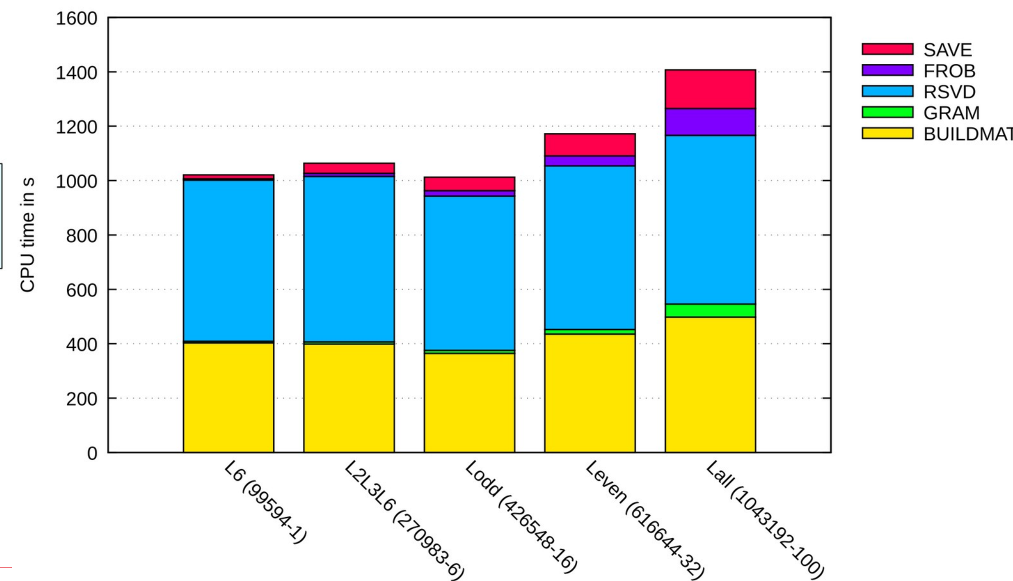
Nuage de points X

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,r} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,r} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,r} \end{pmatrix}$$

Randomized SVD



MDS on 100 Occigen nodes



Accelerated on J. Zay (4 v100/node)

- GEMM of size r * n**2 , n = 1 million
  → x10 faster with GPUs
- QR of size r * n
  → x3 faster with GPUs
- I/O become predominant

Merci. Des questions ?