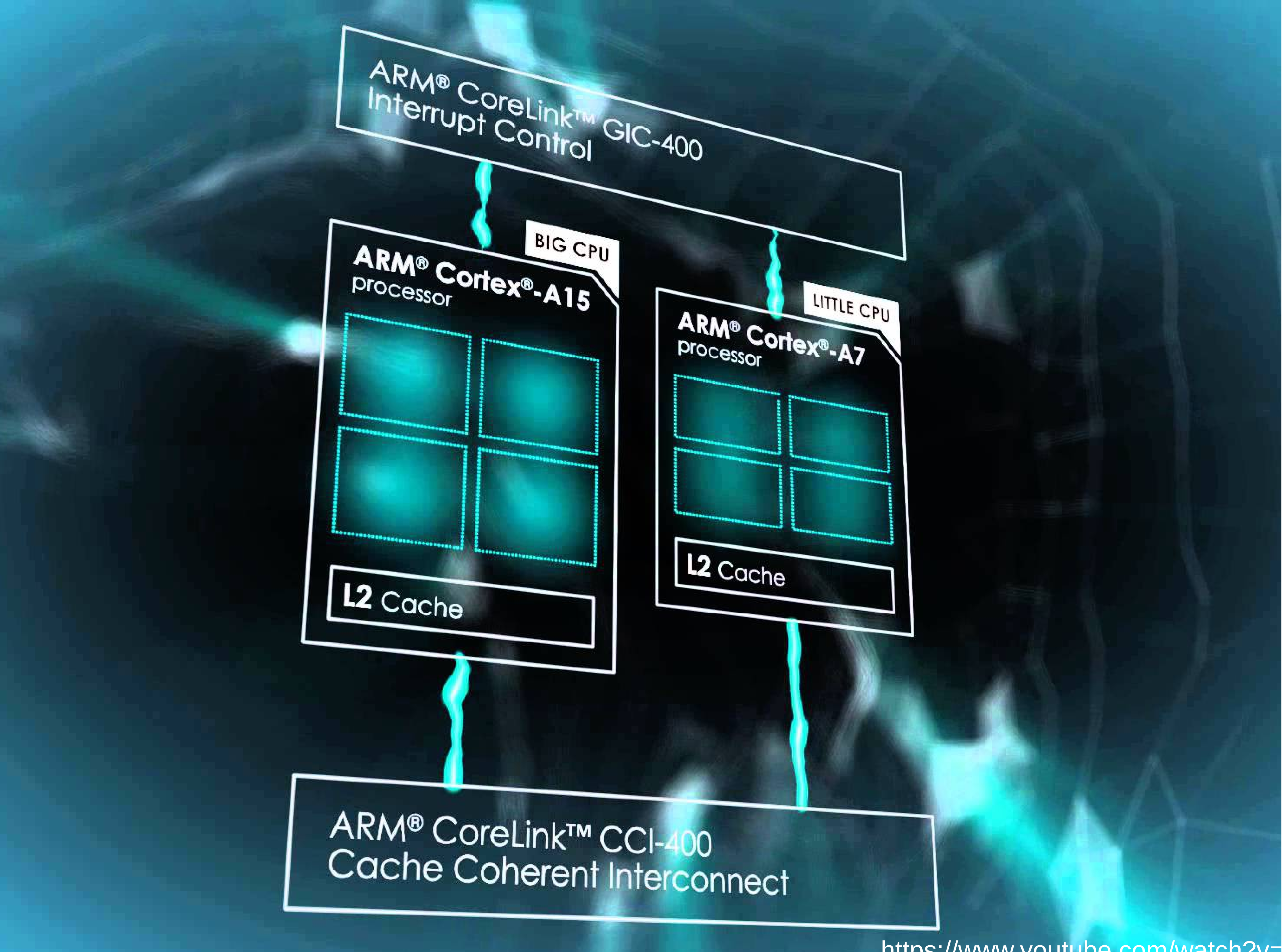
# Refactoring Conventional Task Schedulers to Exploit Asymmetric ARM big.LITTLE Architectures in Dense Linear Algebra

Luis Costero, Francisco D. Igual, Katzalin Olcoz

Sandra Catalán, Rafael Rodríguez-Sánchez, Enrique S. Quintana-Ortí

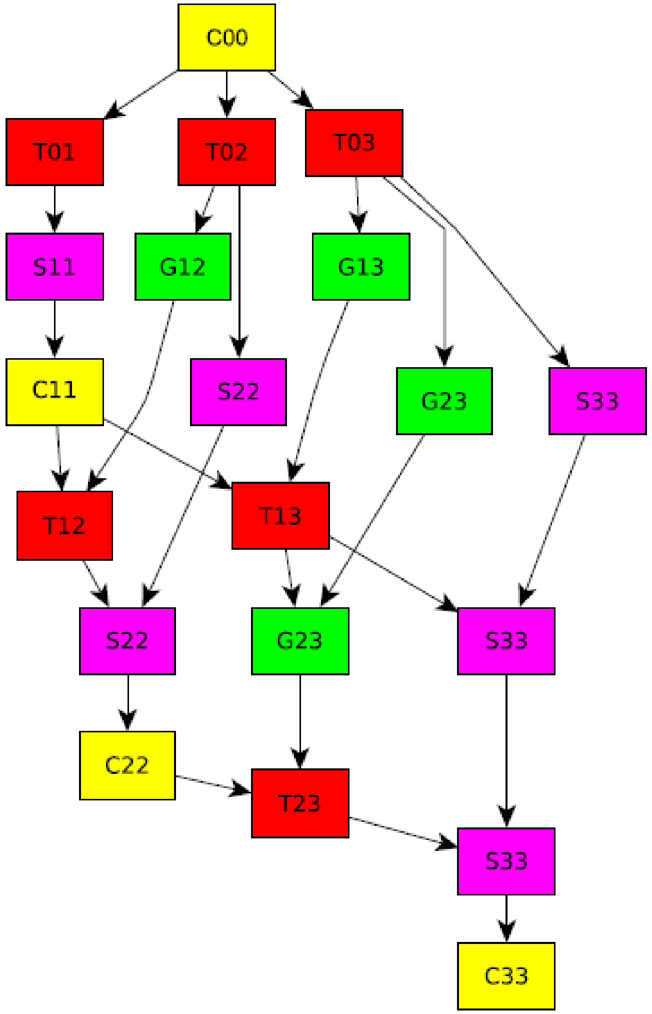
UNIVERSITAT JAUME·I



https://www.youtube.com/watch?v=

# Task parallelism

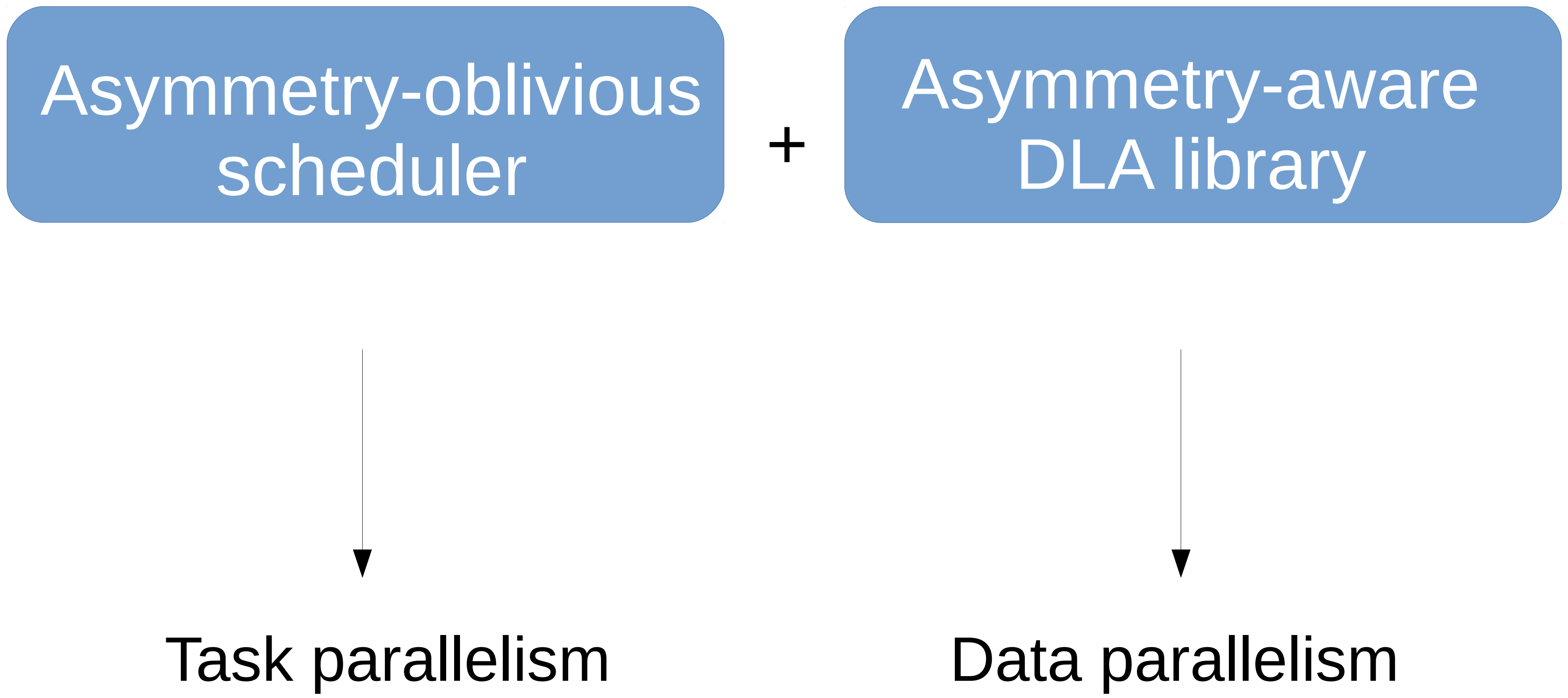
```
void cholesky (double *A[s][s], int b, int s)
{
  for (int k = 0; k < s; k++) {
    // Cholesky factorization (diagonal block)
    po_cholesky (A[k][k], b, b);

    for (int j = k + 1; j < s; j++)
      // Triangular solve
      tr_solve (A[k][k], A[k][j], b, b);

    for (int i = k + 1; i < s; i++) {
      for (int j = i + 1; j < s; j++)
        // Matrix multiplication
        ge_multiply (A[k][i], A[k][j],
                     A[i][j], b, b);
      // Symmetric rank-b update
      sy_update (A[k][i], A[i][i], b, b);
    }

  }
}
```



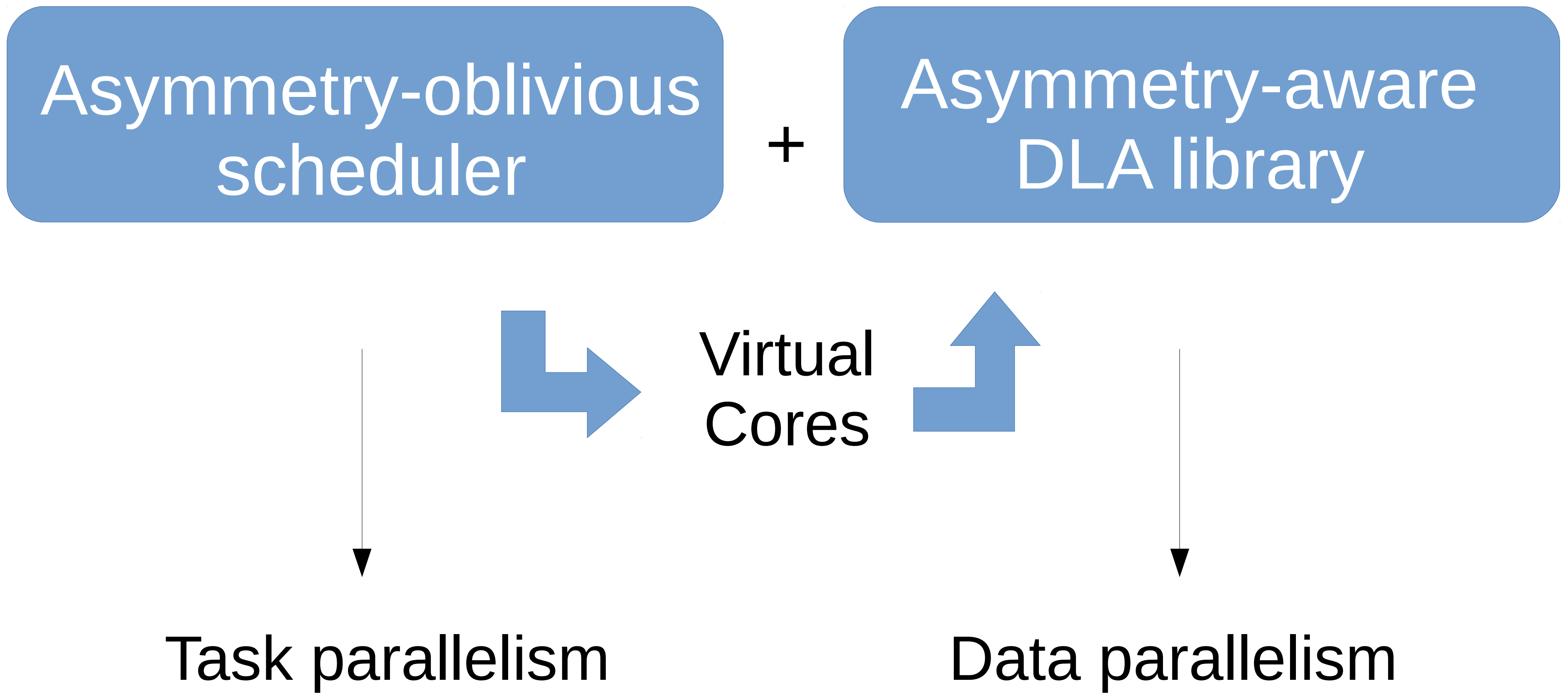
## Contribution

Asymmetry-oblivious scheduler + Asymmetry-aware DLA library

# Contribution

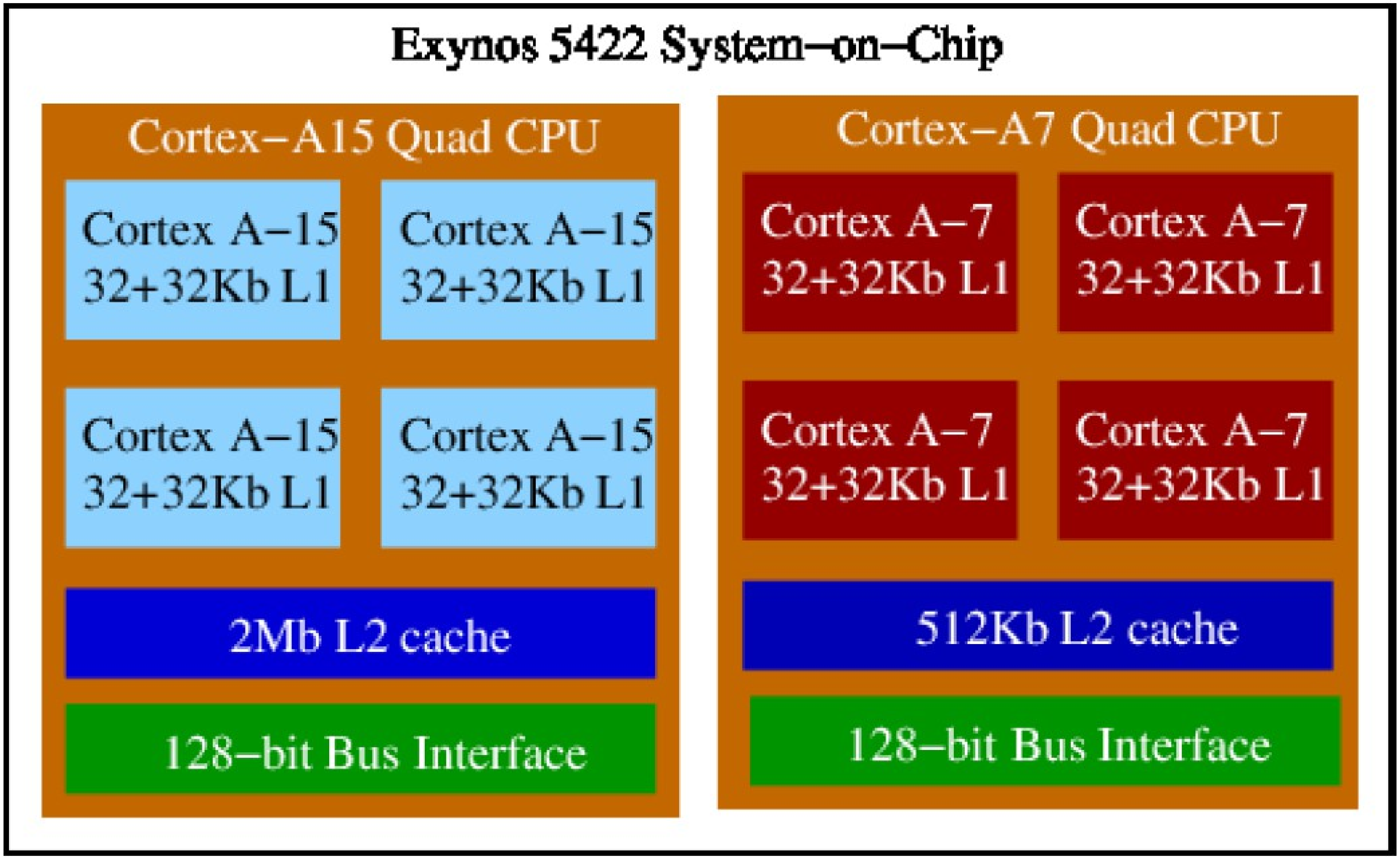


# Contribution

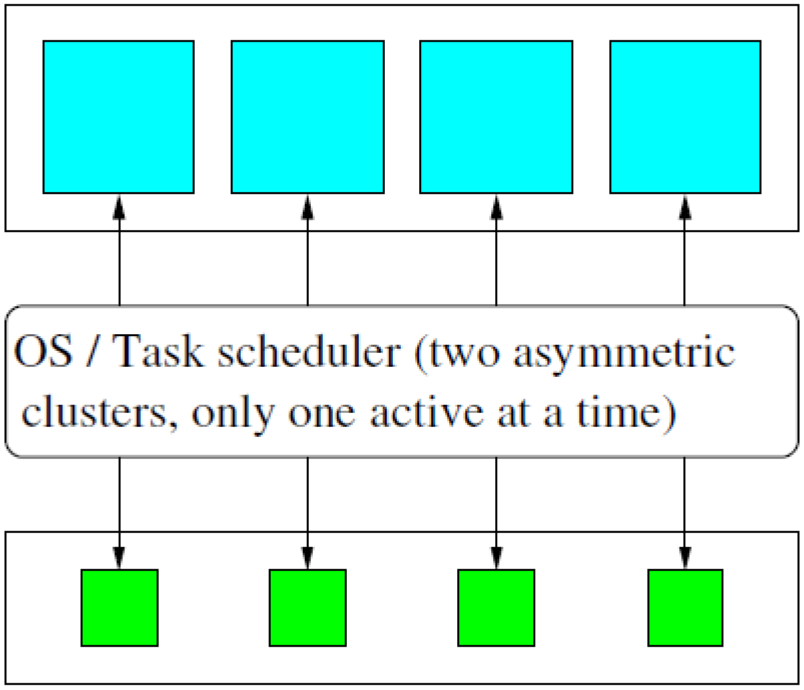


## Software execution models for ARM big.LITTLE

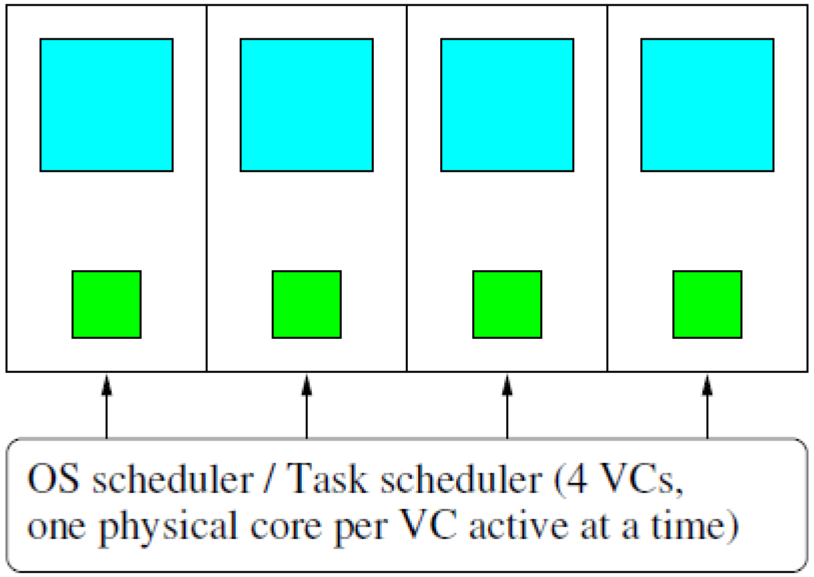
# Target architecture



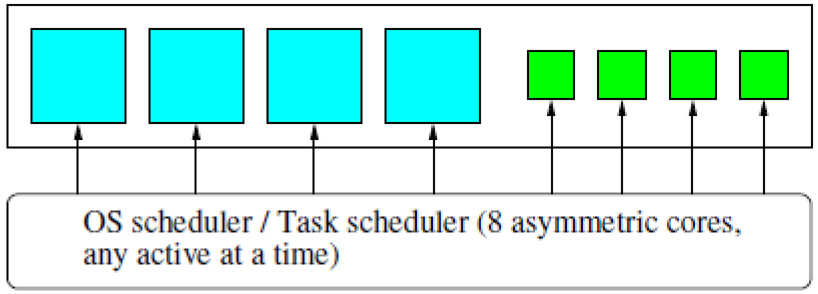
# Execution Models


OS / Task scheduler (two asymmetric clusters, only one active at a time)

Cluster swithching mode


OS scheduler / Task scheduler (4 VCs, one physical core per VC active at a time)

CPU Migration


OS scheduler / Task scheduler (8 asymmetric cores, any active at a time)

Global task scheduling

## Parallel execution of DLA operations on multi-threaded architectures

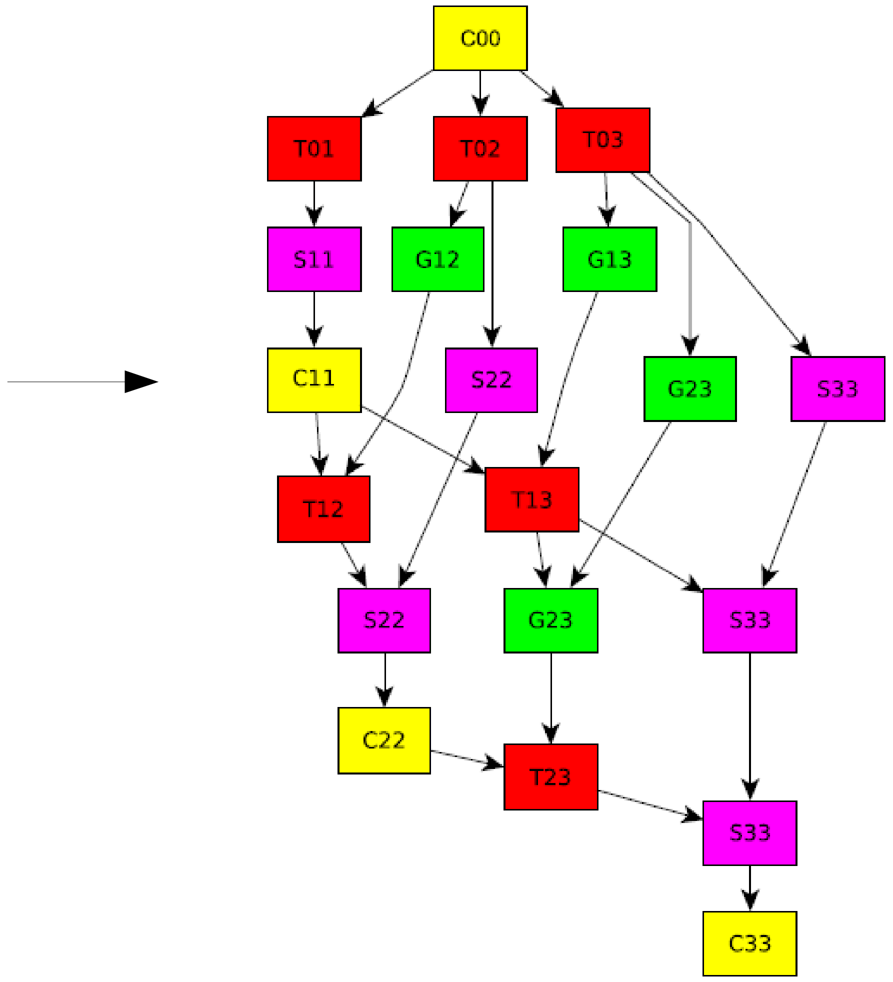
## $A=U^TU$

```
void cholesky (double *A[s][s], int b, int s)
{
  for (int k = 0; k < s; k++) {
     // Cholesky factorization (diagonal block)
     po_cholesky (A[k][k], b, b);

     for (int j = k + 1; j < s; j++)
        // Triangular solve
        tr_solve (A[k][k], A[k][j], b, b);

     for (int i = k + 1; i < s; i++) {
        for (int j = i + 1; j < s; j++)
           // Matrix multiplication
           ge_multiply (A[k][i], A[k][j],
                        A[i][j], b, b);
        // Symmetric rank-b update
        sy_update (A[k][i], A[i][i], b, b);
     }

  }
}
```

# Runtime task scheduling of DLA operations

- Task scheduling for the Cholesky factorization

```
#pragma omp task inout([b][b]A)
void po_cholesky (double *A, int b, int ld)
{
 static int          INFO = 0;
 static const char UP   = 'U';

 // LAPACK Cholesky factorization
 dpotrf (&UP, &b, A, &ld, &INFO);
}

#pragma omp task in([b][b]A) inout([b][b]B)
void tr_solve (double *A, double *B, int b, int ld)
{
 static double       DONE = 1.0;
 static const char LE   = 'L', UP = 'U',
                   TR = 'T',   NU = 'N';

 // BLAS-3 triangular solve
 dtrsm (&LE, &UP, &TR, &NU, &b, &b,
        &DONE, A, &ld, B, &ld);
}

#pragma omp task in([b][b]A, [b][b]B) inout([b][b]C)
void ge_multiply (double *A, double *B, double *C, int b,
     int ld)
{
 static double       DONE = 1.0, DMONE = -1.0;
 static const char TR   = 'T', NT    = 'N';

 // BLAS-3 matrix multiplication
 dgemm (&TR, &NT, &b, &b, &b,
        &DMONE, A, &ld, B, &ld,
        &DONE,  C, &ld);
}

#pragma omp task in([b][b]A) inout([b][b]C)
void sy_update (double *A, double *C, int b, int ld)
{
 static double       DONE = 1.0, DMONE = -1.0;
 static const char UP   = 'U', TR    = 'T';

 // BLAS-3 symmetric rank-b update
 dsyrk (&UP, &TR, &b, &b,
        &DMONE, A, &ld,
        &DONE,  C, &ld);
}
```



## Runtime task scheduling of DLA operations

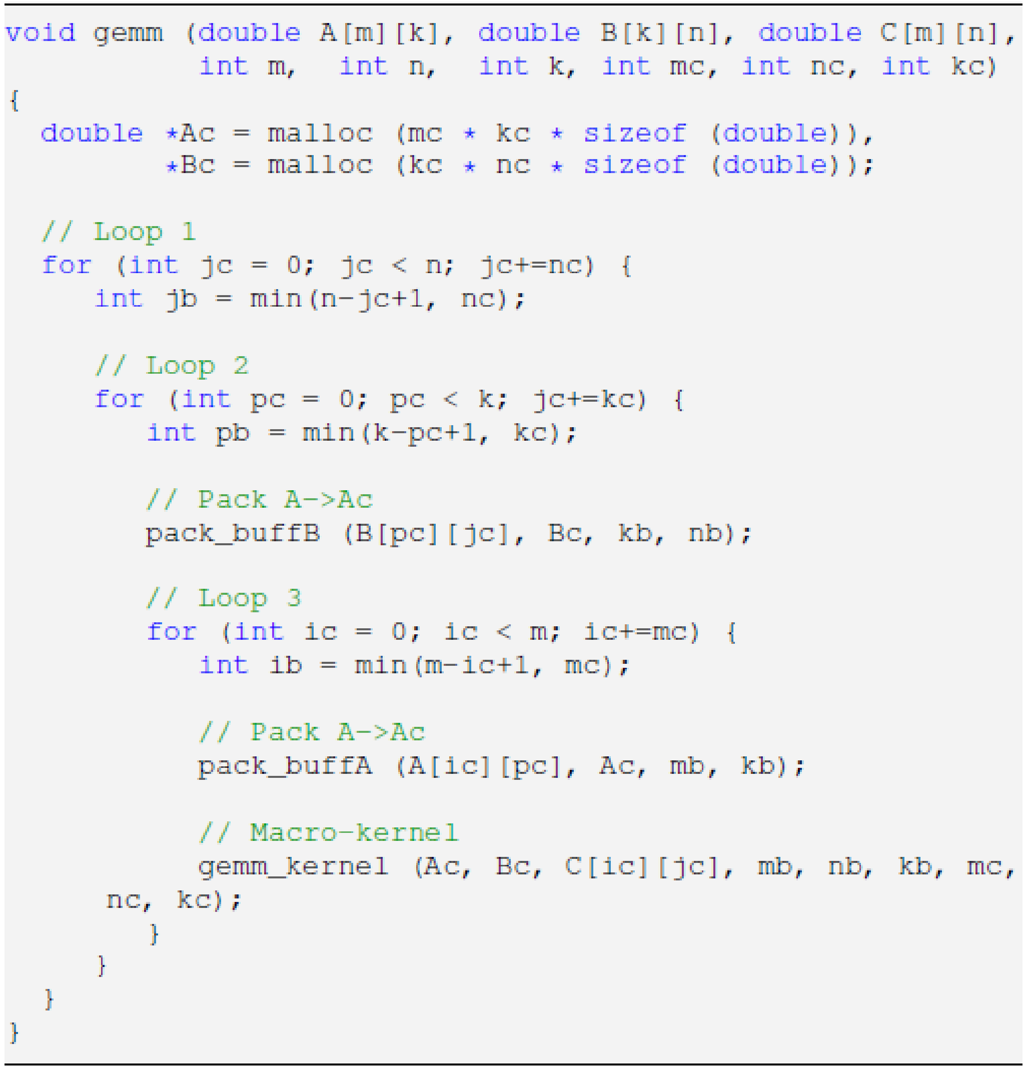
- Task scheduling in heterogeneous architectures
  - The runtime distinguishes between CPU and GPU targets: OmpSs, StarPU, MAGMA, libflame
  - Tasks assigned depending on target properties and specific techniques are applied

## Runtime task scheduling of DLA operations

- Task scheduling in asymmetric architectures
  - Asymmetry-concious runtime: Botlev-OmpSs
  - Critical-aware Task Scheduler policy
  - Each task is mapped to a single core

# Data parallel libraries of BLAS3 kernels

- Multi-threaded implementation of the BLAS-3



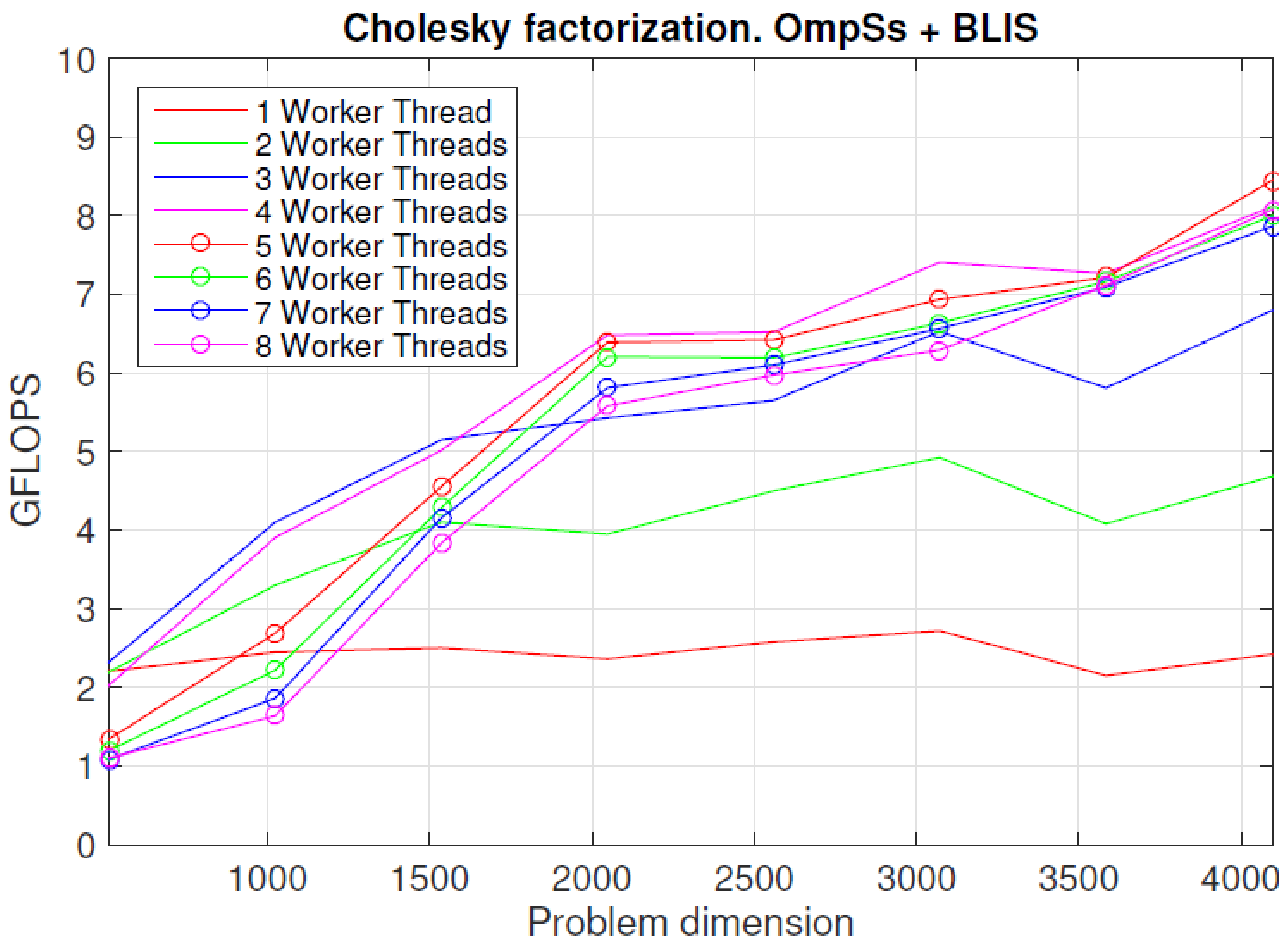
```
void gemm (double A[m][k], double B[k][n], double C[m][n],
           int m,  int n,  int k, int mc, int nc, int kc)
{
  double *Ac = malloc (mc * kc * sizeof (double)),
         *Bc = malloc (kc * nc * sizeof (double));

  // Loop 1
  for (int jc = 0; jc < n; jc+=nc) {
     int jb = min(n-jc+1, nc);

     // Loop 2
     for (int pc = 0; pc < k; jc+=kc) {
        int pb = min(k-pc+1, kc);

        // Pack A->Ac
        pack_buffB (B[pc][jc], Bc, kb, nb);

        // Loop 3
        for (int ic = 0; ic < m; ic+=mc) {
           int ib = min(m-ic+1, mc);

           // Pack A->Ac
           pack_buffA (A[ic][pc], Ac, mb, kb);

           // Macro-kernel
           gemm_kernel (Ac, Bc, C[ic][jc], mb, nb, kb, mc,
      nc, kc);
        }
     }
  }
}
```

# Data parallel libraries of BLAS3 kernels

- Data-parallel libraries for asymmetric architectures:
  - Global Task Scheduling
  - Dynamic workload distribution between the clusters
  - Static workload distribution in a cluster
  - Specific loop strides for each type of core

## Retargeting existing task schedulers to asymmetric architectures

# Evaluation of conventional runtimes on AMPs



# Combining conventional runtimes with asymmetric libraries

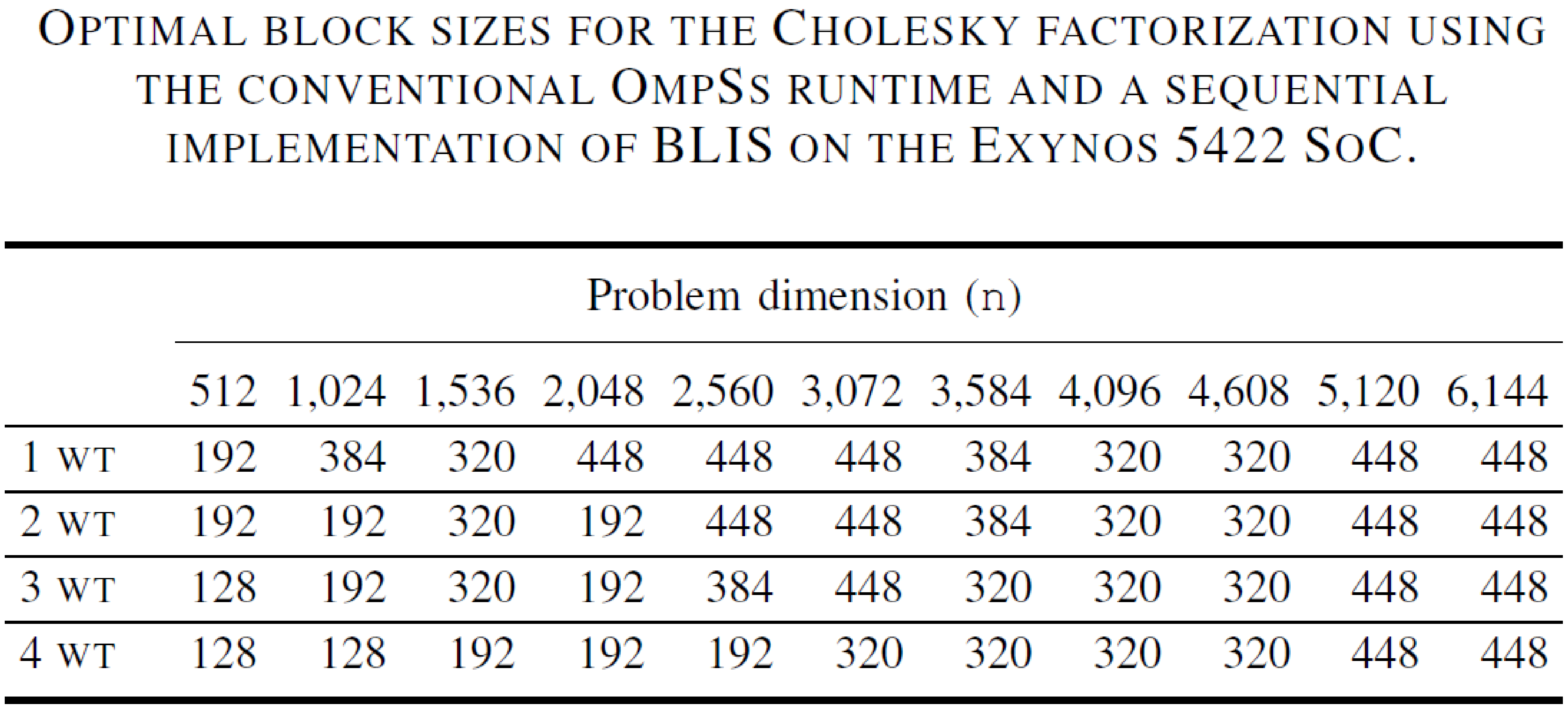
- GTS model (inspired in CPUM)
  - Virtual cores composed of 1A15 + 1A7
  - Both cores are active simultaneously
- Parallelism:
  - Task-level: symmetric runtime
  - Data-level: asymmetric library

## Combining conventional runtimes with asymmetric libraries

- Comparison with other approaches:
  - ✓ Any conventional task scheduler will work transparently with no special modifications
  - ✓ Any improvement in the runtime will impact the performance on an AMP
  - ✓ Any improvement in the asymmetry-aware library will impact the performace on an AMP
  - ✗ Need of a tuned asymmetry-aware DLA library

## Experimental results

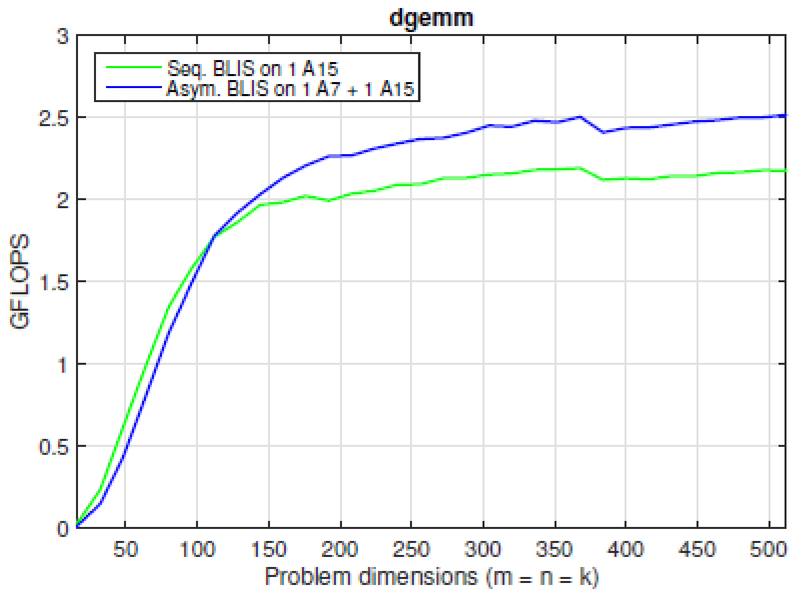
# Performance evaluation of the asymmetric BLIS

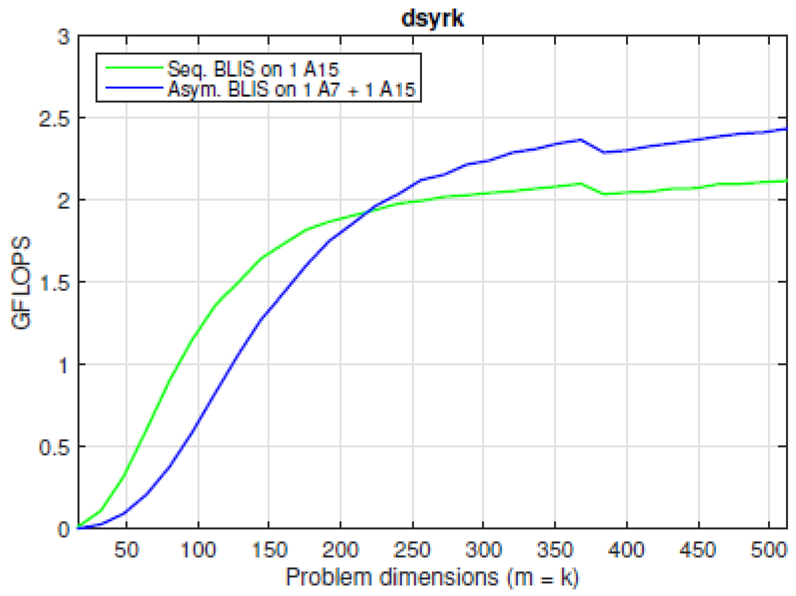


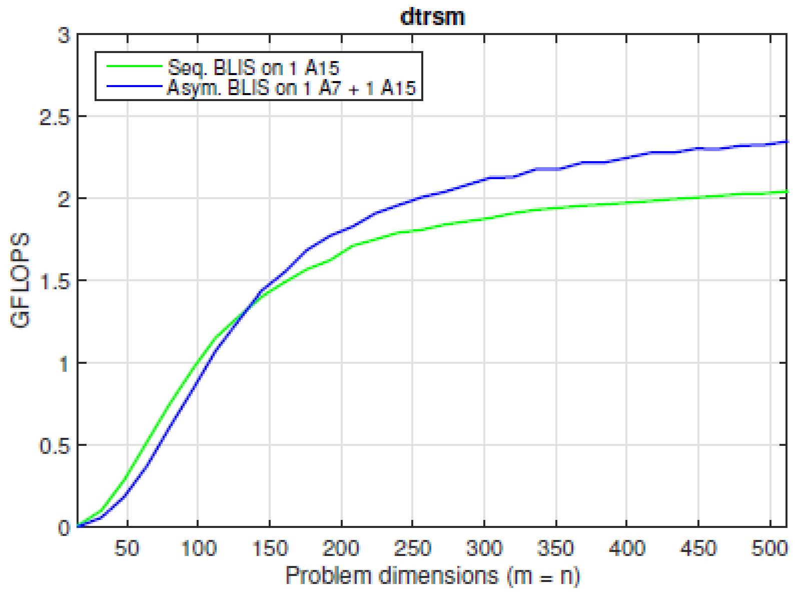
OPTIMAL BLOCK SIZES FOR THE CHOLESKY FACTORIZATION USING THE CONVENTIONAL OMPSS RUNTIME AND A SEQUENTIAL IMPLEMENTATION OF BLIS ON THE EXYNOS 5422 SOC.

| | Problem dimension (n) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 512 | 1,024 | 1,536 | 2,048 | 2,560 | 3,072 | 3,584 | 4,096 | 4,608 | 5,120 | 6,144 |
| 1 WT | 192 | 384 | 320 | 448 | 448 | 448 | 384 | 320 | 320 | 448 | 448 |
| 2 WT | 192 | 192 | 320 | 192 | 448 | 448 | 384 | 320 | 320 | 448 | 448 |
| 3 WT | 128 | 192 | 320 | 192 | 384 | 448 | 320 | 320 | 320 | 448 | 448 |
| 4 WT | 128 | 128 | 192 | 192 | 192 | 320 | 320 | 320 | 320 | 448 | 448 |

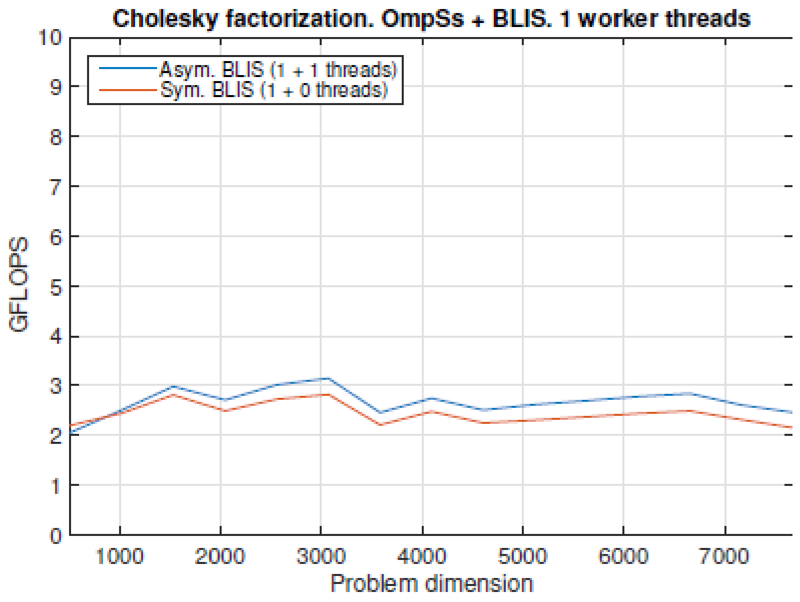
# Performance evaluation of the asymmetric BLIS

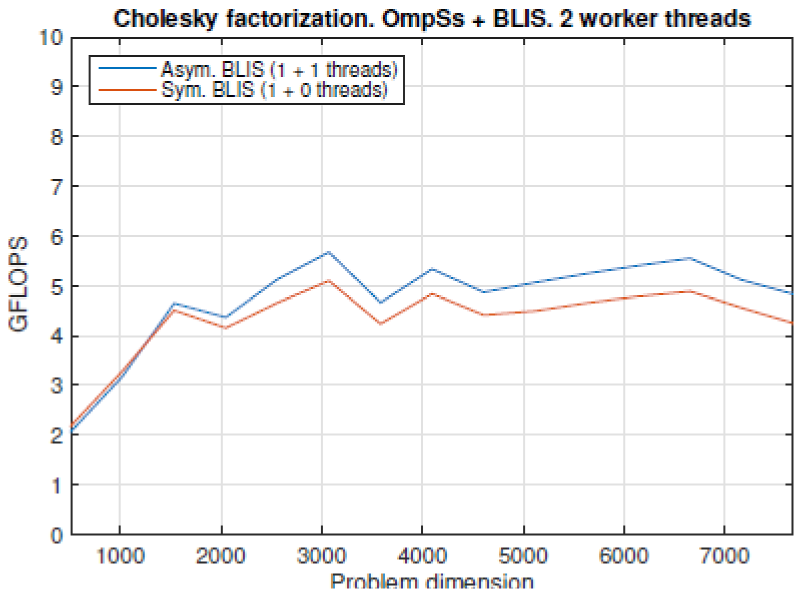


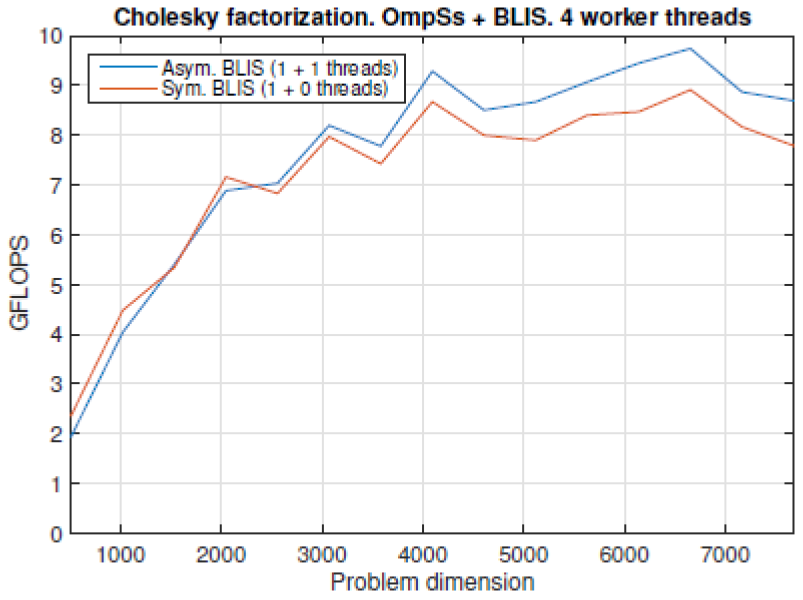




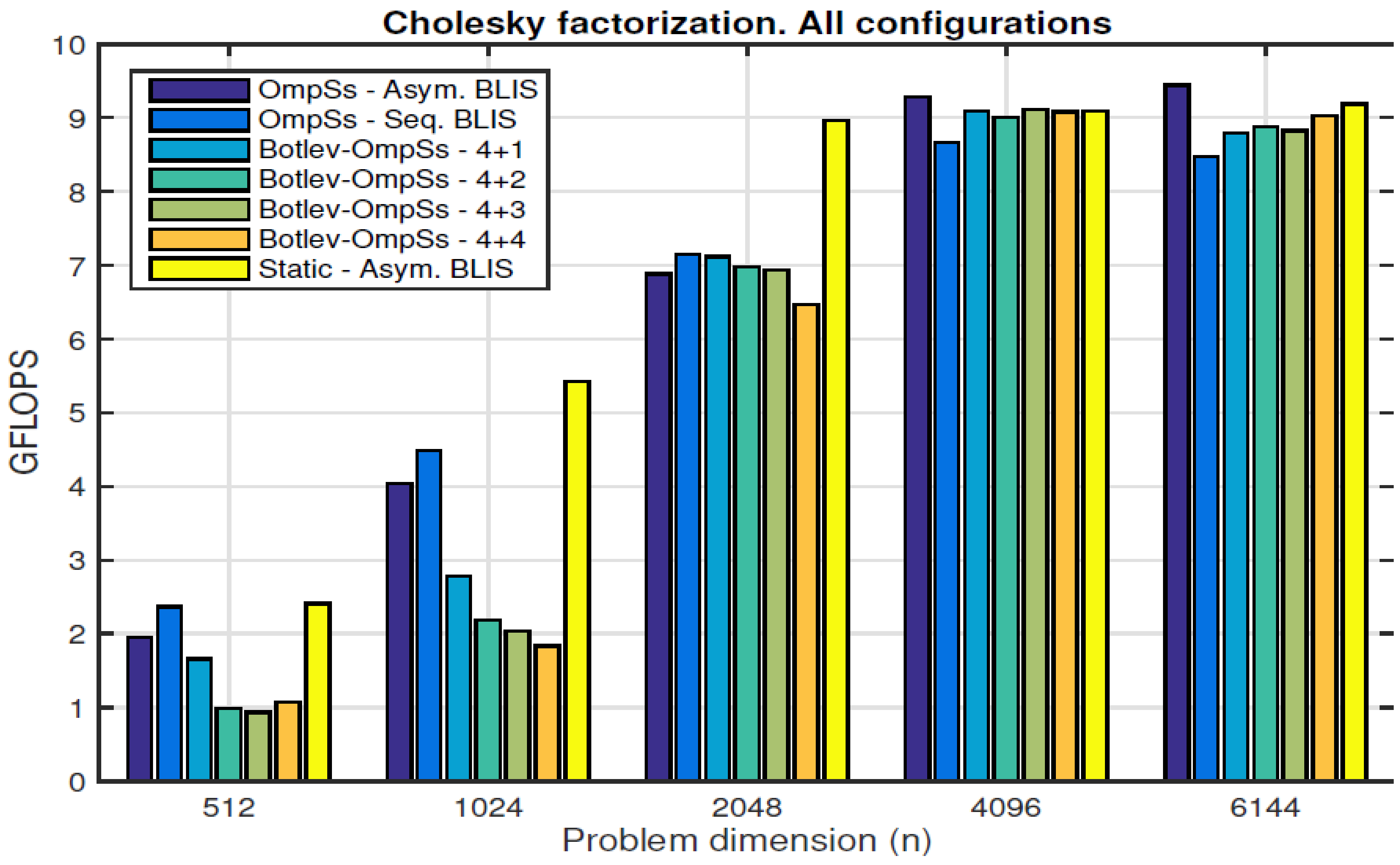
# Integration of the asymmetric BLIS in a conventional task scheduler







# Performance comparison versus asymmetry-aware task scheduler



## Conclusions

## In this work...

- Task-parallelism + Data-parallelism on AMPs
- Reuse of existing task schedulers.
- Competitive with asymmetry-aware schedulers

Thank you