Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Hybridizing MPI and tasking: The MPI+OmpSs experience

Jesús Labarta

BSC – CS Dept. Director

Russian Supercomputing Days

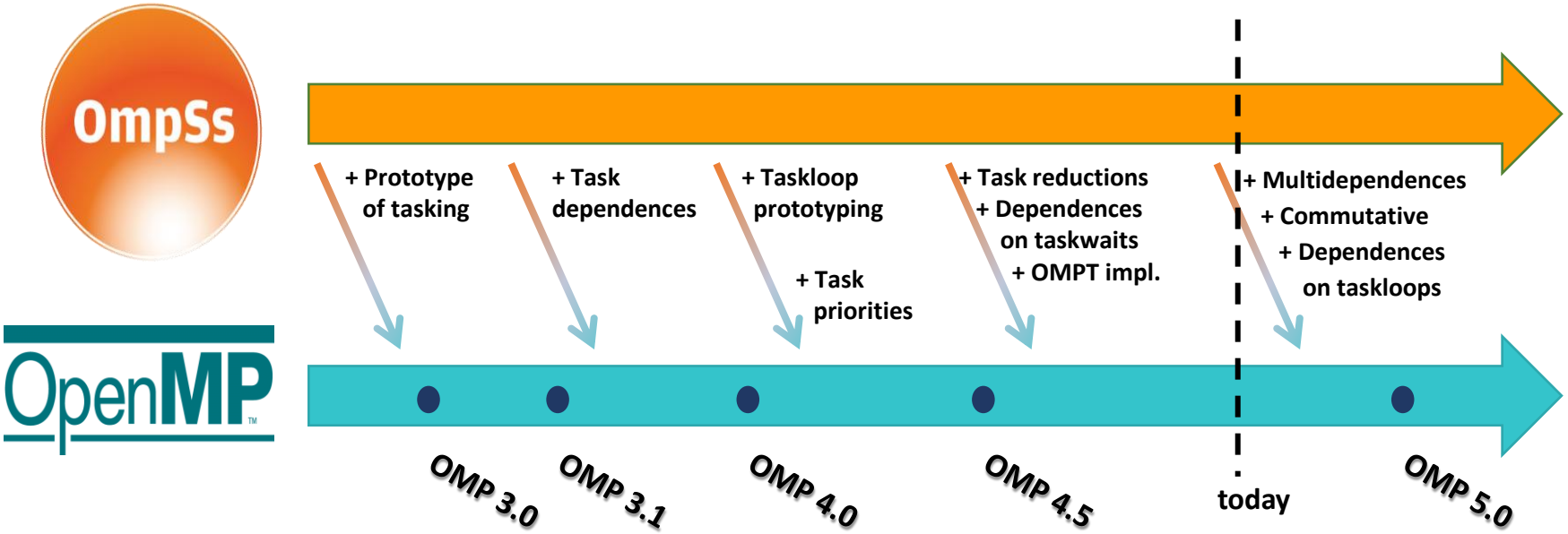
Moscow, September 25th, 2017

MPI + X

- Why hybrid?
  - MPI is here to stay
  - A lot of HPC applications already written in MPI
  - MPI scales well to tens/hundreds of thousands of nodes
  - MPI + X:
    - MPI exploits inter-node parallelism while X can exploit node parallelism
- Standards
  - X = OpenMP
- This talk:
  - X = OmpSs (~OpenMP)
  - Mechanisms & Mentality

OmpSs

- A forerunner for OpenMP



OmpSs in one slide

Color code: OpenMP, influenced OpenMP, pushing, not yet

- Minimalist set of concepts …

```
#pragma omp task [ in (array_spec, l_values...)] [ out (...)] [ inout (…, v[neigh[j]], j=0;n)]) \
                 [ concurrent (…)] [commutative(...)] [ priority(P) ] [ label(...) ] \
                 [ shared(...)][private(...)][firstprivate(...)][default(...)][untied] \
                 [final(expr)][if (expression)] \
                 [reduction(identifier : list)] \
                 [resources(…)]
 {code block or function}
```

```
#pragma omp taskwait [ { in | out | inout } (...) ] [noflush]
```

```
#pragma omp taskloop  [grainsize(…) ] [num_tasks(…) [nogroup]   [ in (...)] [reduction(identifier : list)]
  {for_loop}
```

```
#pragma omp target device ({ smp | opencl | cuda })     \
        [ implements ( function_name )]                 \
        [ copy_deps | no_copy_deps ] [ copy_in ( array_spec ,...)] [ copy_out (...)] [ copy_inout (...)] } \
        [ndrange (dim, …)] [shmem(...) ]
```

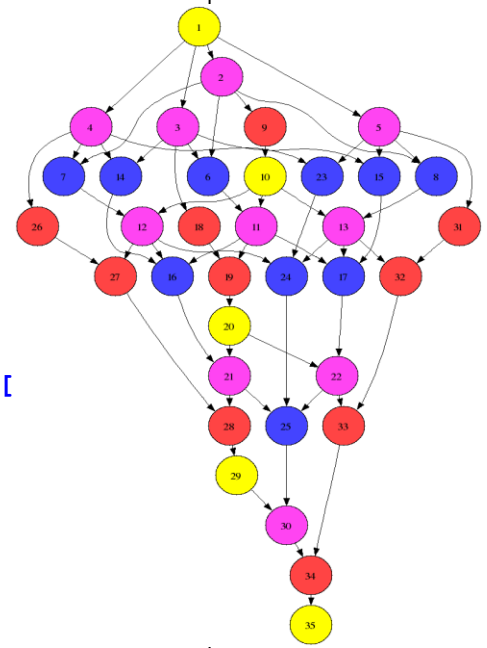

A. Duran, et al, "Extending the OpenMP Tasking Model to Allow Dependent Tasks" IWOMP 2008, LNCS & IJPP

E. Ayguade, et al, "A Proposal to Extend the OpenMP Tasking Model for Heterogeneous Architectures" IWOMP 2009 & IJPP

OmpSs/ OpenMP tasks

- Out of order execution honoring dependences

```
void Cholesky(int NT, float *A[NT][NT] ) {
   for (int k=0; k<NT; k++) {
      #pragma omp task inout ([TS][TS]A[k][k])
      spotrf (A[k][k], TS) ;
      for (int i=k+1; i<NT; i++) {
        #pragma omp task in(([TS][TS]A[k][k])) \
                         inout ([TS][TS]A[k][i])
        strsm (A[k][k], A[k][i], TS);
      }
      for (int i=k+1; i<NT; i++) {
        for (j=k+1; j<i; j++) {
          #pragma omp task in([TS][TS]A[k][i]),\
                           in([TS][TS]A[k][j])  inout ([TS][TS]A[
          sgemm( A[k][i], A[k][j], A[j][i], TS);
        }
        #pragma omp task in ([TS][TS]A[k][i]) \
                         inout([TS][TS]A[i][i])
        ssyrk (A[k][i], A[i][i], TS);
      }
   }
}
```

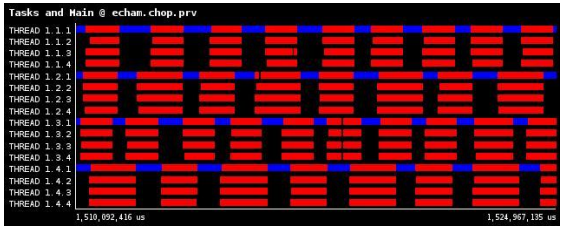


MPI + OpenMP/OmpSs

- Communication can be taskified or not
  - If not taskified:
    - main program has to stall till data to be sent and the reception buffers are available.
    - still communication performed by main thread can overlap with previously generated tasks if they do not refer to communicated variables
    - Benefits can be obtained by hybrid approach **if** MPI imbalance increases with process count and it is possible to balance at OpenMP level
  - If taskified
    - it can be instantiated and program can proceed generating work.
    - Tasks calling MPI will execute out of order with respect to any other task just honoring local dependences within the process
- Other approaches
  - Threads invoking MPI ☹

Not taskifying communications

- Parallelize individual computation phases
  - The Amdahl's law challenge !!!!
    - OmpSs still an opportunity to fight it
    - A chance for lazy programmers



ECHAM

MAXW-DGTD

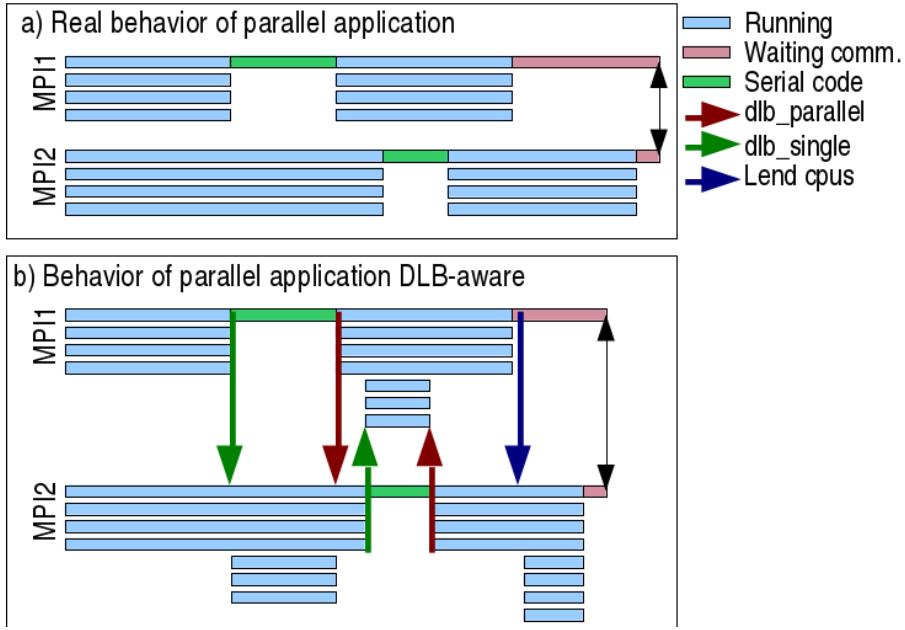
GROMACS@SMPSs

- still some potential communication/computation overlap
  - If previously generated tasks do not refer to communicated variables can delay the taskwait clause

Dynamic Load Balancing

- Automatically achieved by the runtime
  - Load balance within node
  - Fine grain.
  - Complementary to application level load balance.
  - **Leverage OmpSs malleability**

- DLB Mechanisms
  - User level Run time Library (DLB)
  - Detection of process needs
    - Intercepting runtime calls
      - Blocking
      - Detection of thread level concurrency
    - Request/release API
  - Coordinating processes within node
    - Through a shared memory region
    - Explicit pinning of threads and handoff scheduling (Fighting the Linux kernel)
  - Within and across apps



"LeWI: A Runtime Balancing Algorithm for Nested Parallelism". M.Garcia et al. ICPP09

Taskifying communications

- Control flow can instantiate communication tasks and proceed generating work
  - Potential overlap between iterations !!

- Tasks calling MPI will execute **out of order** with respect to computation and **communication tasks**

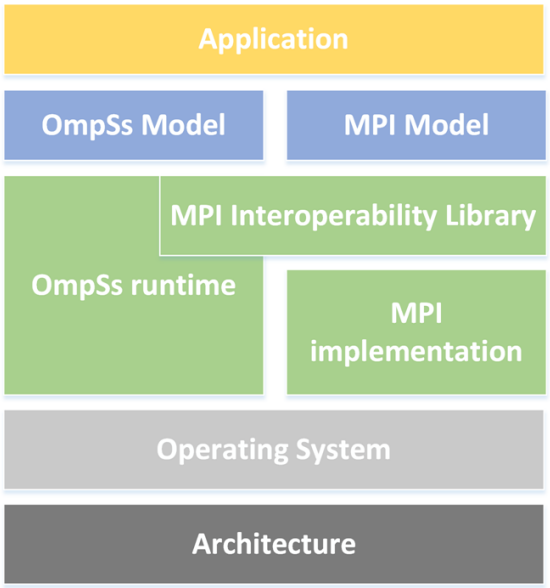
- Communicating Tasks:
  - Single MPI communication
    - Could be done just modifying mpi.h
  - Burst of MPI communications (+ some small but potentially critical computation between them)
    - Less overhead
    - Typically the inter process synchronization cost is paid in the first MPI call within the task

Taskifying communications

- Issues to consider
  - Possibly several concurrent MPI calls → thread safe MPI
    - Might not be available or really concurrent in some installations
  - Internal MPI state and semantics is also shared state → may cause dependences
  - MPI tasks waste cores if communication partner delays or long transfer times
    - Less problem as more and more cores per node become available
  - Reordering of MPI calls + blocking calls + limited number of cores → potential to introduce deadlock
    - → need to control order of communication tasks

Deadlock potential when taskifying communications

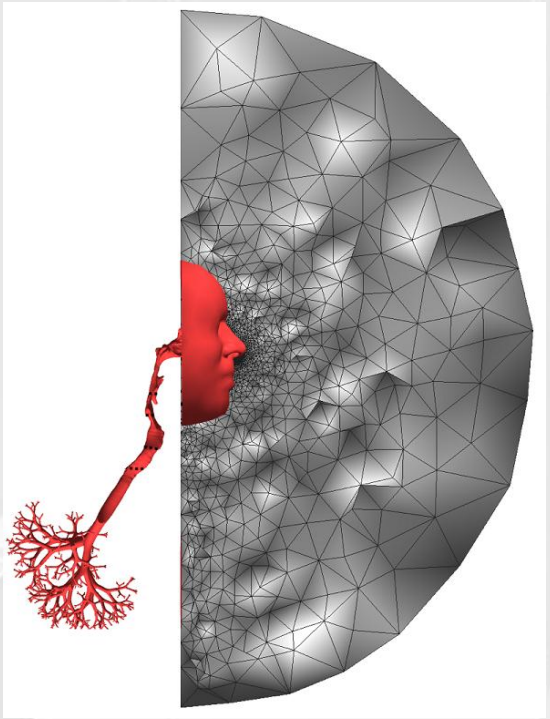
- Approaches to handle
  - Algorithmic structure may not expose loop and thus be deadlock free
  - Enforce in order execution of communicating tasks
    - Dependences
    - Inout(sentinel) on all communication tasks
    - Serialization of communication can have an impact on performance
  - Virtualize communication resource. Allow infinite communication tasks
    - MPI-OmpSs interoperability library
      - Blocking MPI calls → nonblocking
      - Block task in NANOS
      - Poll completion
      - Reactivate task when communication completes
    - Also addresses issues of wasted cores
      - Interaction with task granularity



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

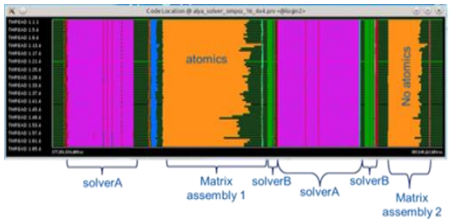
EXCELENCIA SEVERO OCHOA

ALYA

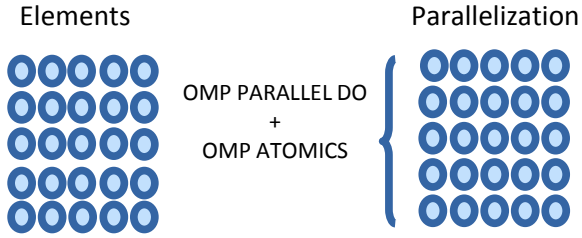


Alya: Parallelization of FE codes

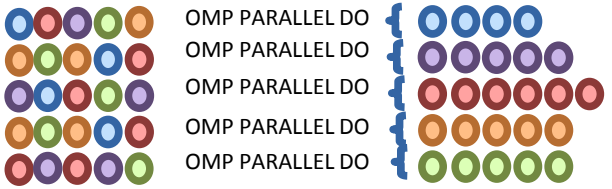
- Interaction Load balance and IPC



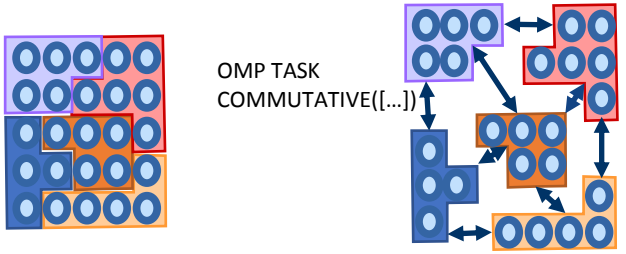
- Reductions with indirect accesses on large arrays
  - No coloring



  - Coloring

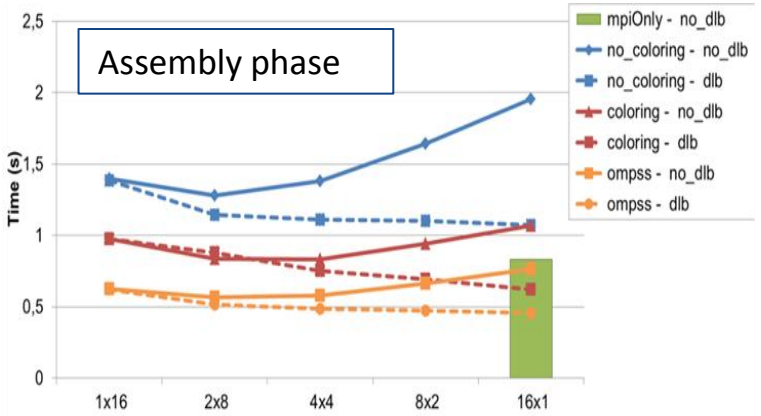


  - Commutative Multidependences
    (OmpSs feature to be hopefully included in OpenMP)

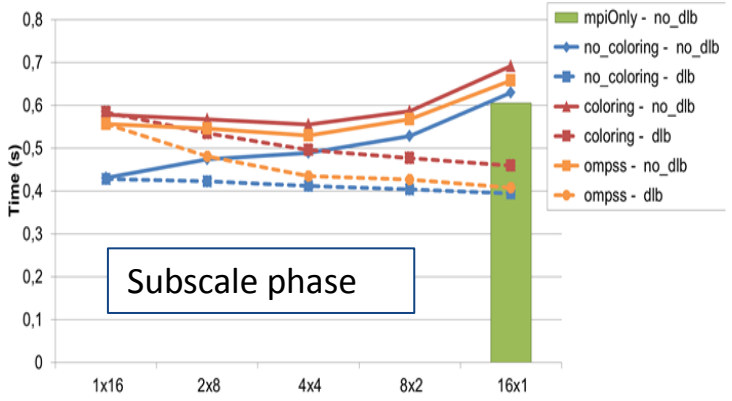


Alya: Taskification and DLB

- Towards throughput computing
  - Malleability (tasks) + DLB → flat lines
- DLB helps in all cases
  - Even more in the bad ones ☺
- Side effects
  - Hybrid MPI+OmpSs Nx1 can be better than pure MPI !!!
  - Nx1 + DLB: hope for lazy programmers

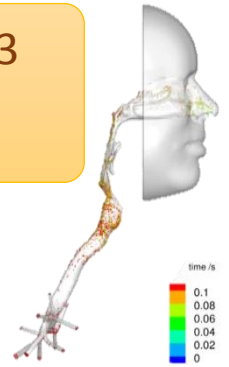


16 nodes x P processes/node x T threads/process

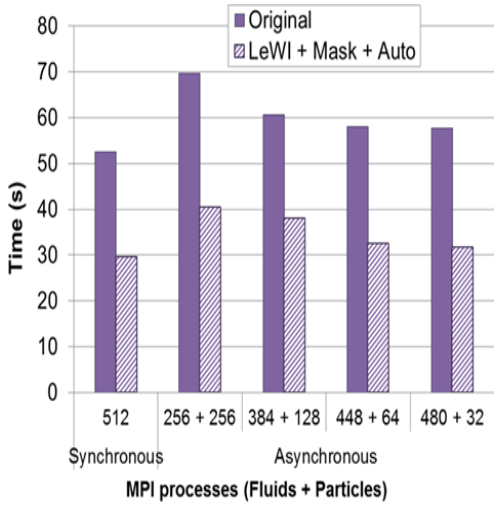


Alya: Coupled codes

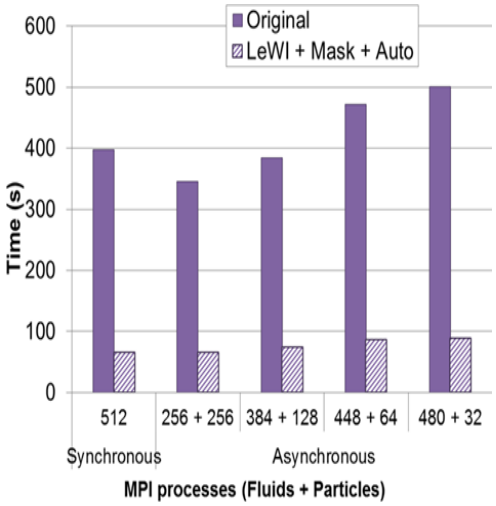
- Marenostrum3
  - 32 nodes
    - 512 cores



- Observations
  - Big impact configuration and kind of coupling in original version
  - Important improvement with DLB-LeWI in all the cases
  - Almost constant performance independent of configuration and kind of coupling



Fluid dominated



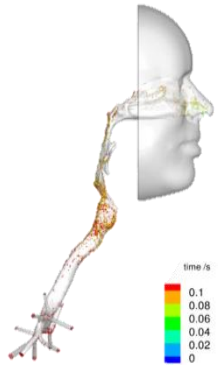
Particle dominated

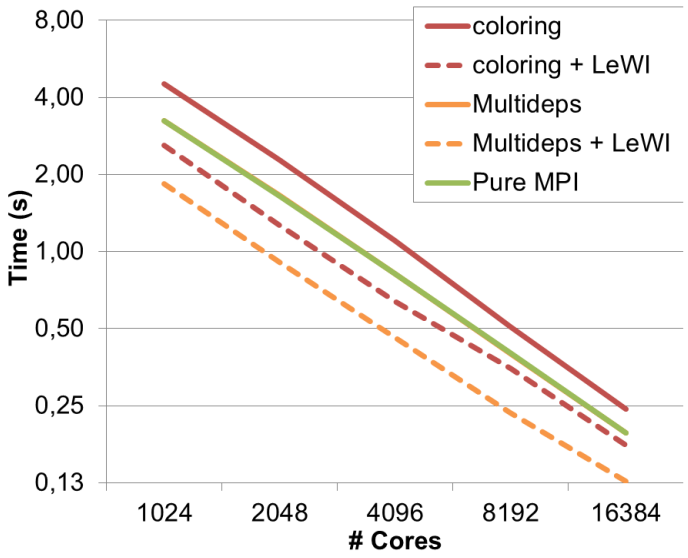
Fluid

Particle

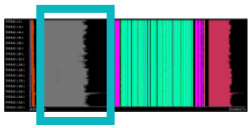
Alya: Scaling

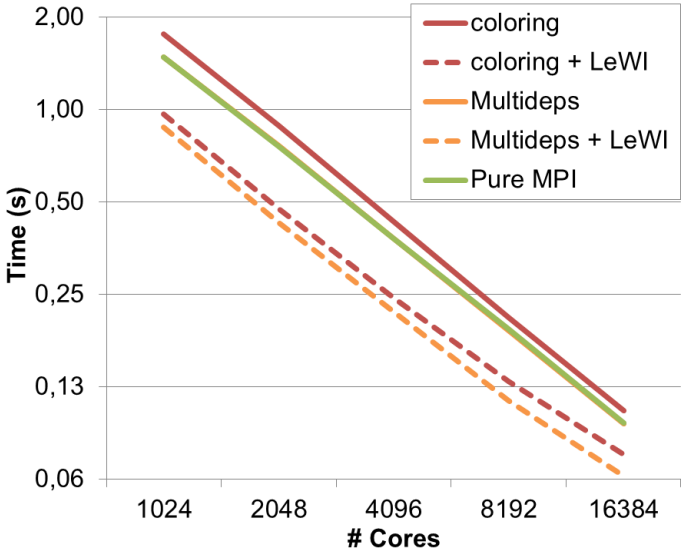
- Observations
  - DLB-LeWI Scaling up to 16k cores
  - DLB-LeWI shows a maintained grain from 1k to 16k cores
  - DLB-LeWI can manage fine granularities



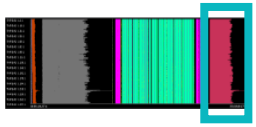


Matrix assembly





Subgrid scale



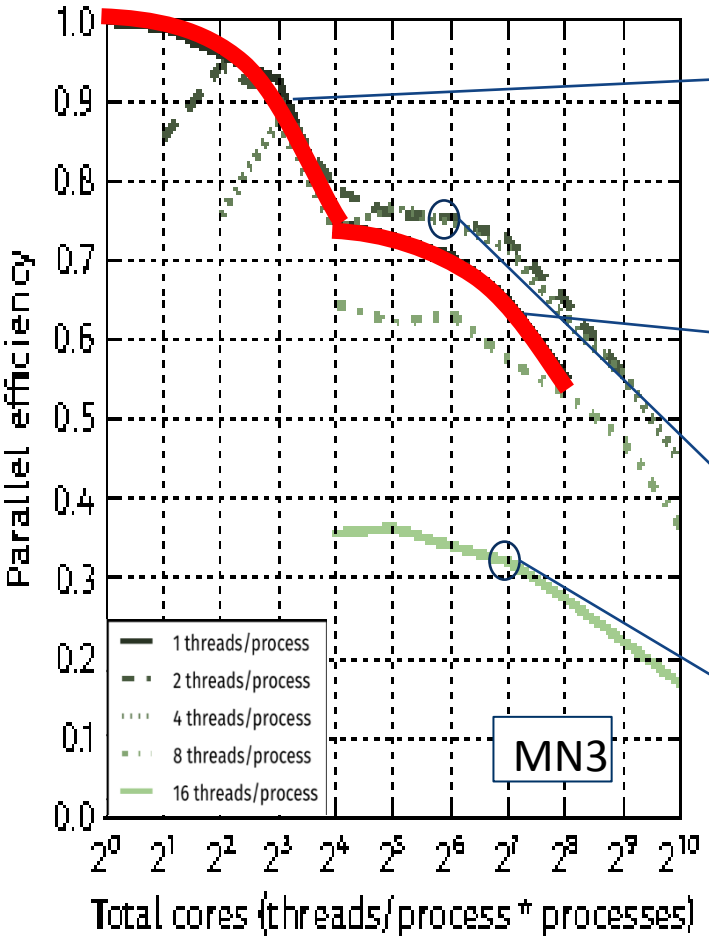
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

NTCHEM

Ntchem

- Original: Hybrid MPI + OpenMP (@MN3)



Not fully populated node → system activates TurboBoost increasing frequency from 2.6 to 3.1GHz

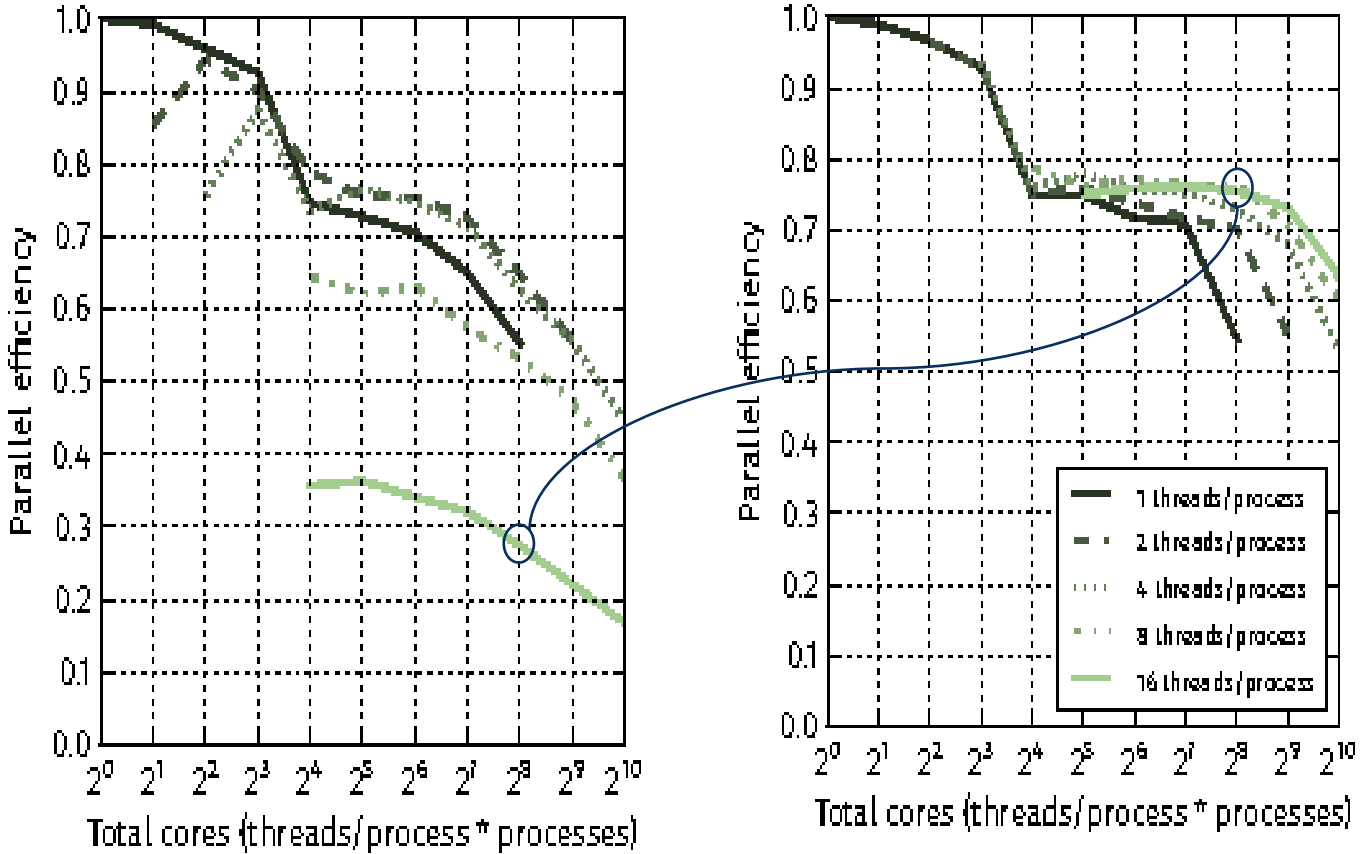
Load imbalance
Global
Serialization
Noise

Some gain @ low threading count

High overhead, fine granularity @ large threading count

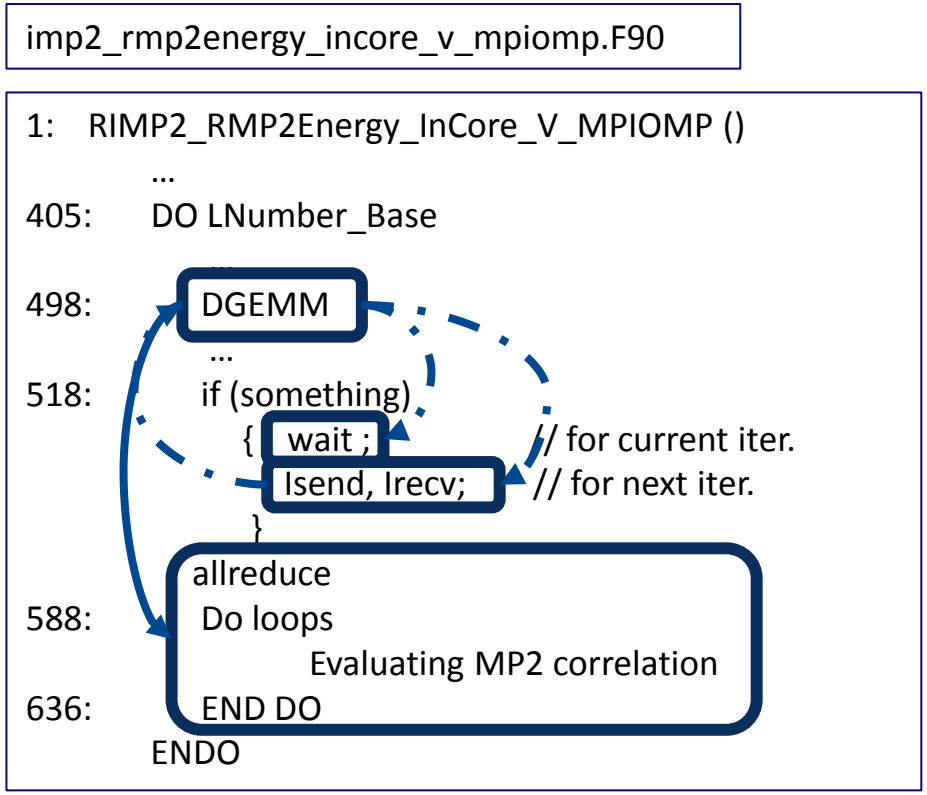
NTChem: OmpSs taskification

- Simpler code, better performance !!
  - Sufficient task granularity
  - Communication computation overlap
  - DLB needed at very large scale?



NTChem: OmpSs taskification

- Taskify
  - Serial DGEMMs.
    - Coarser granularity than original OpenMP
  - Reduction loops
    - Not parallelized in original?
    - Serial task
  - Communications
    - Overlap, but fixed schedule in original source
- Outcome
  - Possible with limited understanding of global application
  - Happen to be fairly independent



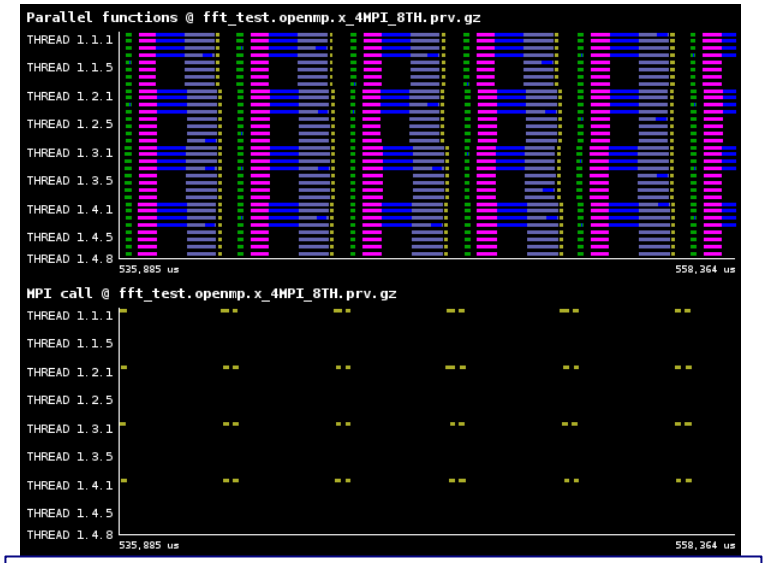
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

FFTLib

FFTLib

- Miniapp for part of Quantum Espreso
  - Many FFT independent iterations performed in series.
    - Collective operations
    - Some additional computation.
- Default MPI+OpenMP
  - OpenMP within each local FFT
- Hybrid MPI+OpenMP Double Buffered
  - Manual allocation, explicit overlap

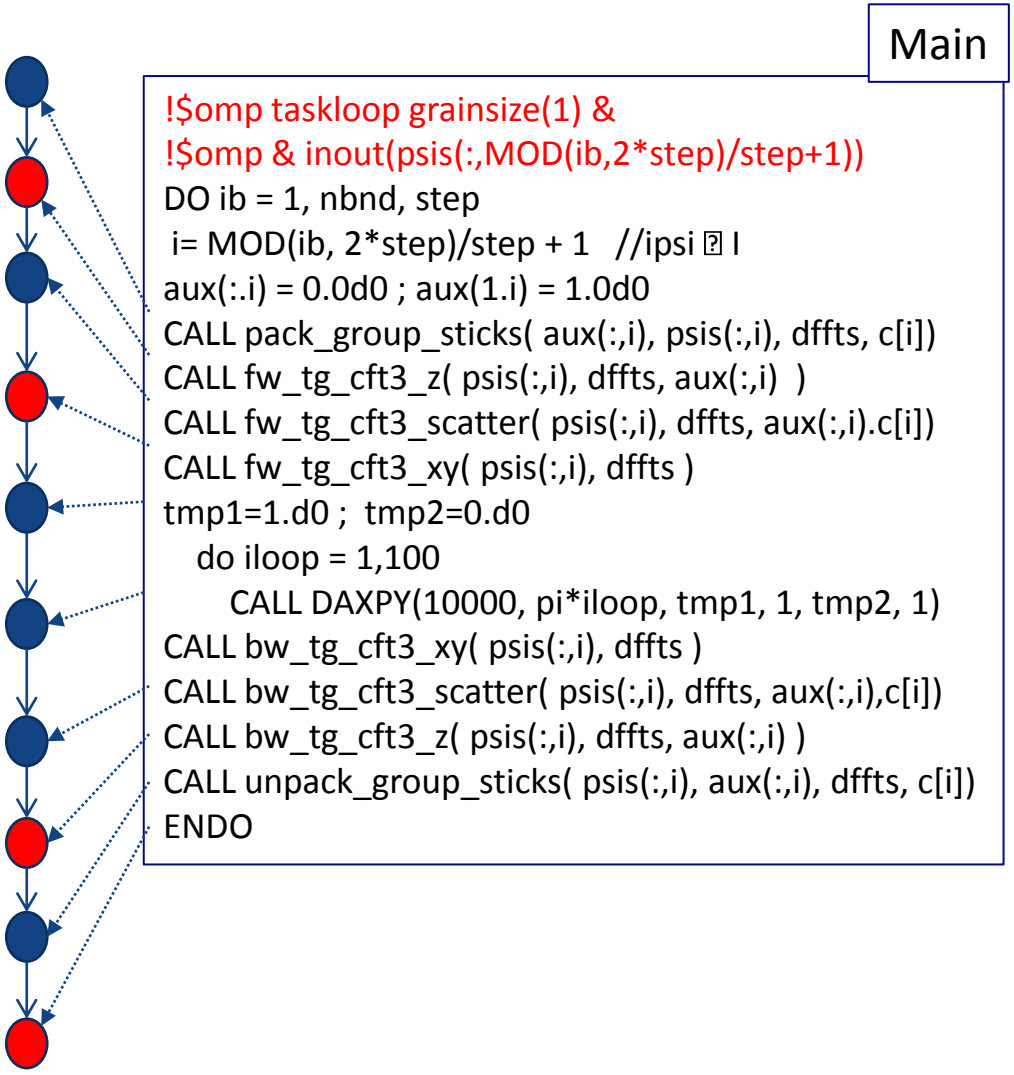


OMP_NUM_THREADS=8
mpirun -np 4 FFTlib -ecutwfc 80 -alat 10 -nbnd 1024 -ntg 4

- Other versions: Complex refactoring of the code to redistribute data and execute different FFTs on separate communicators

Structure

- Syntactic structure
  - Flow dependence chain within iteration
  - Anti dependence across iterations
    - Vector expansion *aux, tmp*
  - Tasks performing collective ops
    - Decouple them → communicator expansion (MPI_COMM_DUP)
  - Expansion can be made proportional to actual need/capability to exploit (i.e. #threads)



Structure

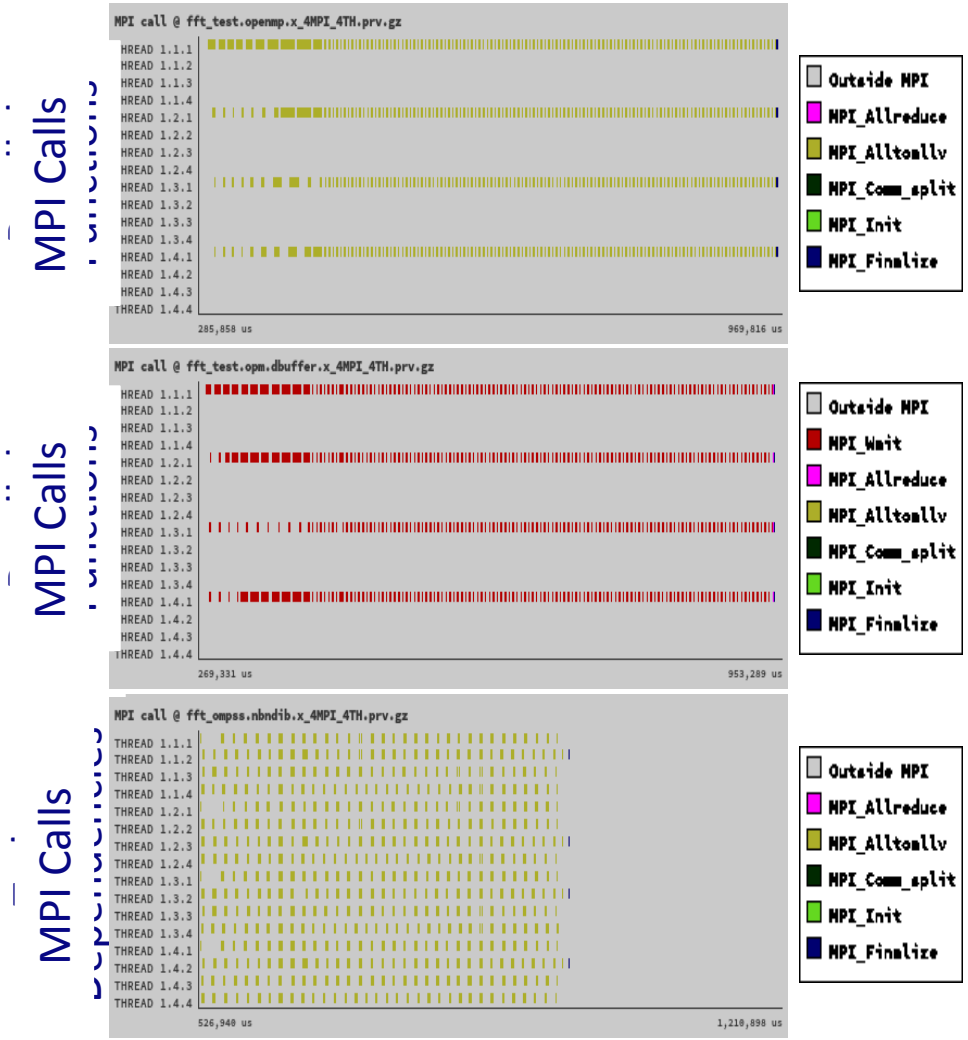
- Syntactic structure (approx.)
  - Nesting
  - Taskloops within fft_scalar.FFTW.f90
  - 2 functions:
    - cft_2xy
    - cft_2z

cft_2xy

```
SUBROUTINE cft_2xy(r, nzl, nx, ny, ldx, ldy, isign, pl2ix)
…
 IF( isign < 0 ) THEN
      tscale = 1.0_DP / ( nx * ny )
      !$omp taskloop num_threads(2) private(offset, i) label(x_stick)
      DO i=1,nzl
        offset = 1+ ((i-1)*(ldx*ldy))
        CALL FFT_X_STICK_SINGLE( fw_plan(1,ip), r(offset), … )
      END DO

      !$omp taskloop num_threads(2)) private(j, i, k) label(y_stick)
      do i = 1, nx
       do k = 1, nzl
        IF( dofft( i ) ) THEN
           j = i + ldx*ldy * ( k - 1 )
           call FFT_Y_STICK(fw_plan(2,ip), r(j), …)
        END IF
       end do
      end do

      r = r * tscale
….
```

Resulting executions (4x4)

- Default MPI + OpenMP
- Default MPI + OpenMP (DOUBLE_BUFFER)
- MPI + OmpSs



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Conclusion

Recommendations

- Parallelization approach
  - Top – down
  - Nesting
  - Use tasks dependences and lookahead to overlap heterogeneous tasks
  - Elimination of antidependences
    - Privatization
    - (Vector) expansion: x2, x3, …

- Communications
  - Expansion of communicators
  - Do not rely on MPI ordering guarantees, use tags
  - Try to avoid nonblocking calls, do not believe you can place wait better than a node level task dependence mechanism

Recommendations

- Libraries
  - DLB
  - MPI – OmpSs/OpenMP interoperability

- OmpSs / OpenMP (co-design wishes)
  - Dependences in taskloops
  - Dynamic task granularity

Conclusion

- Hybrid MPI + OpenMP/OmpSs has a lot of potential
  - Much more than typically exploited
  - Propagates OmpSs asynchornous out of order features to global program behavior
  - Enables …
- …the real parallel programming revolution …
  - … is in the mindset of programmers
  - … is to rely on the runtime and system
  - … is to change mindset for the latency age to the throughput age !!!!
- … towards exascale
- … and before !!!!

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thank you

Jesus.labarta@bsc.es

07/07/2017

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thank you

Jesus.labarta@bsc.es

07/07/2017

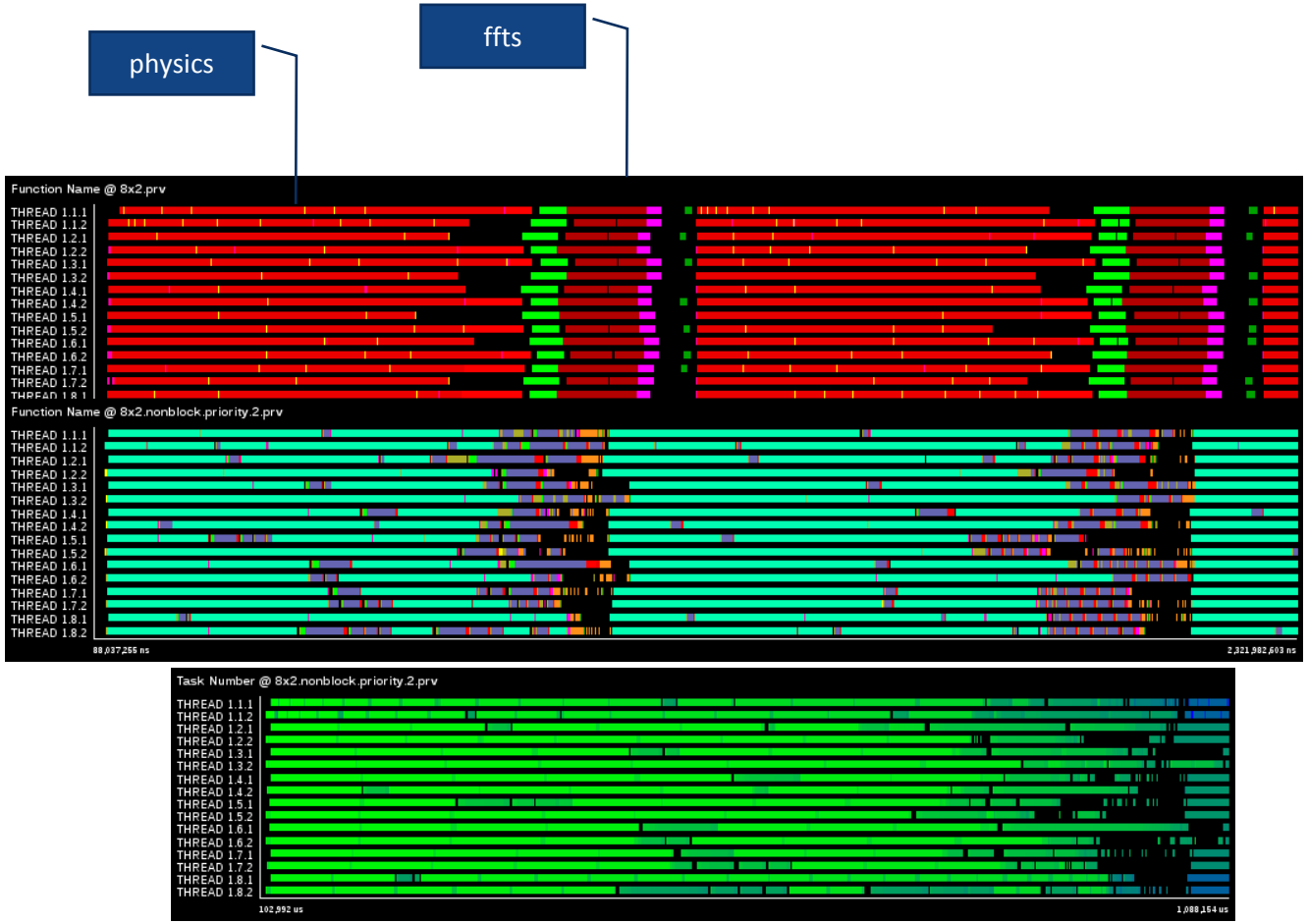
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

IFS Kernel

MPI-OmpSs Interoperability Library

- IFS weather code kernel (ECMWF)



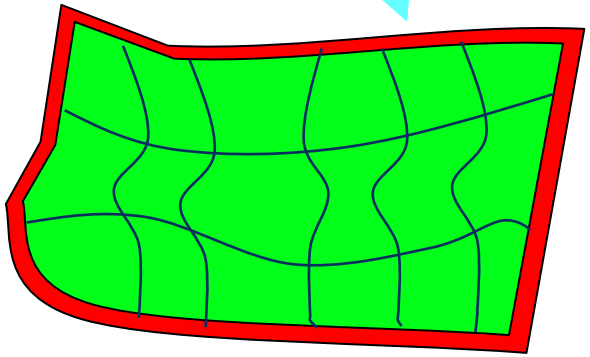
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

SPECFEM3D

MPI/OmpSs code structure

Domain: boundary and inner elements



```
for iter
   Compute(red)
   exchange(red, neighbors_list)
   Compute (green)
   #pragma omp taskwait
   Update (red and green)
```

Few tasks

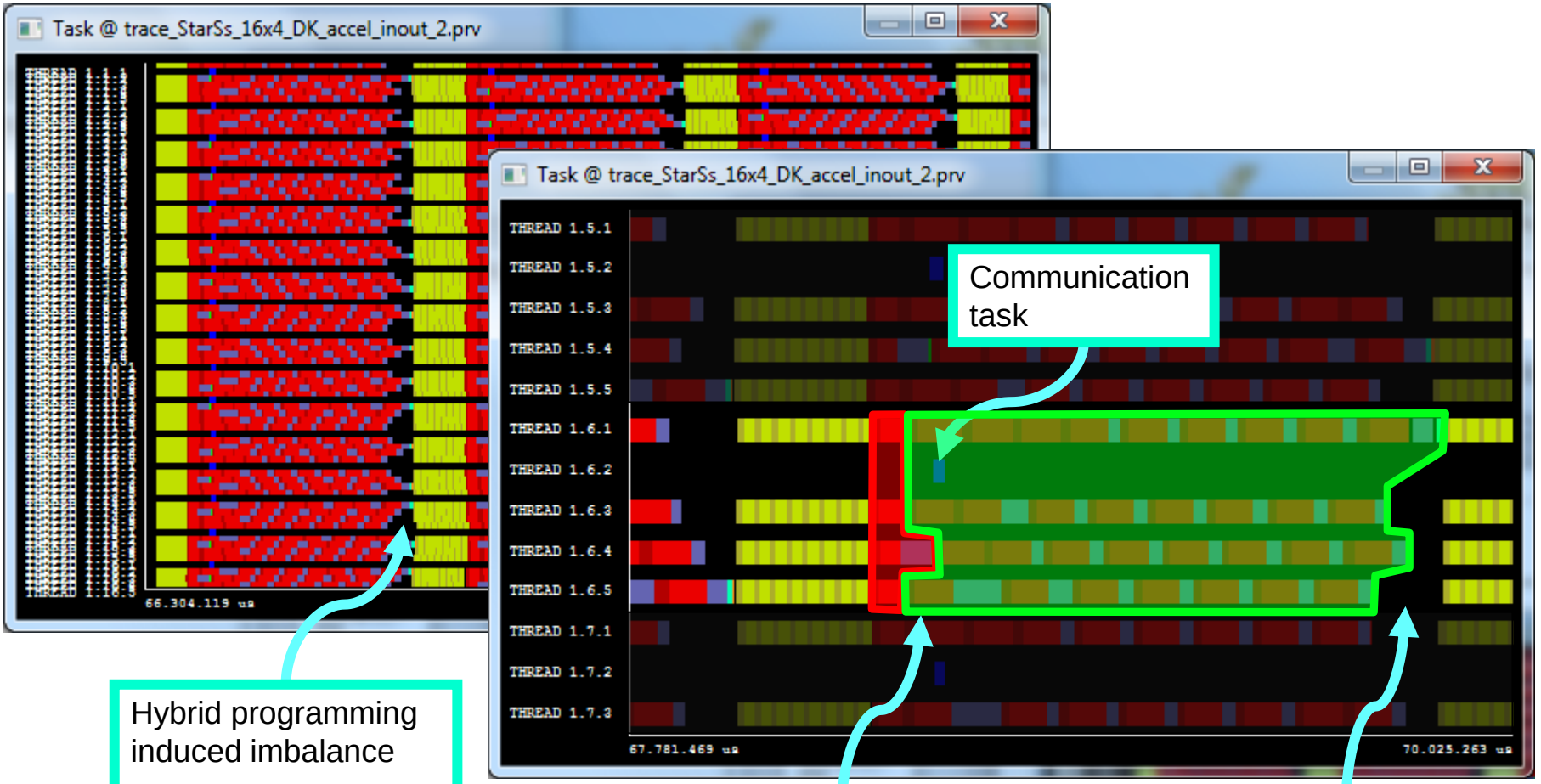
Dependence

One task

Many tasks

Node numbering

Trace analyses



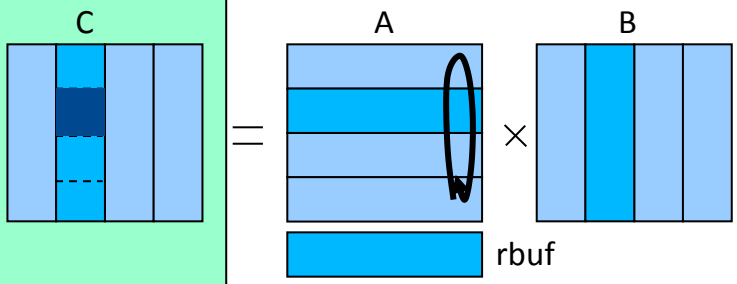
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

MxM

MxM @ MPI

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {

   mxm( m, n, a, B, (double (*)[n])&C[i][0]);

   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;

}


free (orig_rbuf);
```

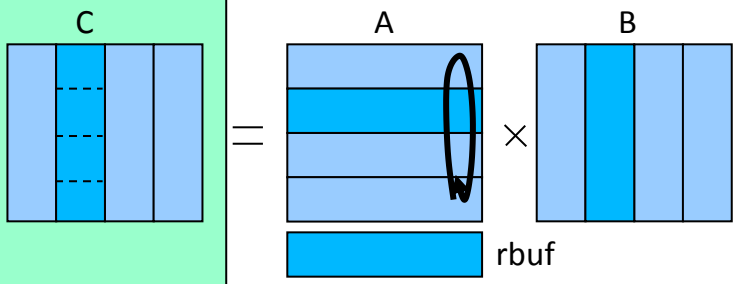



```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j;
   char tr = 't';
   dgemm(&tr, &tr, &m, &n, &n, &alpha, a, &m, b, &n, &beta, c, &n);
}
```

```
void callSendRecv(int m, int n,
                  double (*a)[m], int down,
                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```

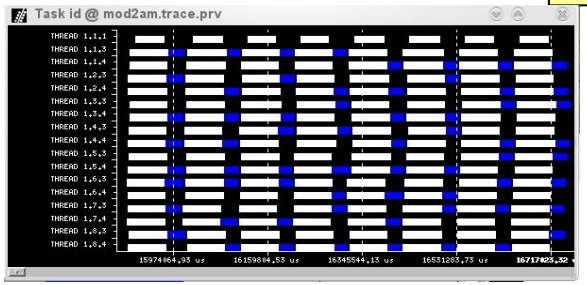
MxM @ MPI + OmpSs: MPI calls not taskified

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {

   mxm( m, n, a, B, (double (*)[n])&C[i][0]);

   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;

}


free (orig_rbuf);
```



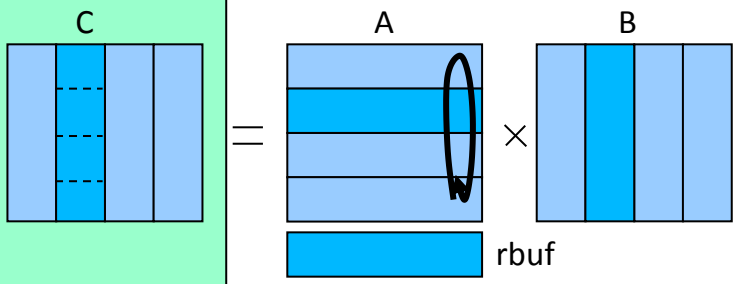

```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j, l=1;
   char tr = 't';
   for (i=0; i<n; i++) {
      #pragma omp task
      dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &m, b, &n, &beta, c, &n);
      c+=n;
      a+=m;
      b+=n;
   }
   #pragma omp taskwait
}
```

```
void callSendRecv(int m, int n,
                  double (*a)[m], int down,
                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```



MxM @ MPI + OmpSs: MPI calls not taskified

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {

   mxm( m, n, a, B, (double (*)[n])&C[i][0]);

   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;
   #pragma omp taskwait
}


free (orig_rbuf);
```



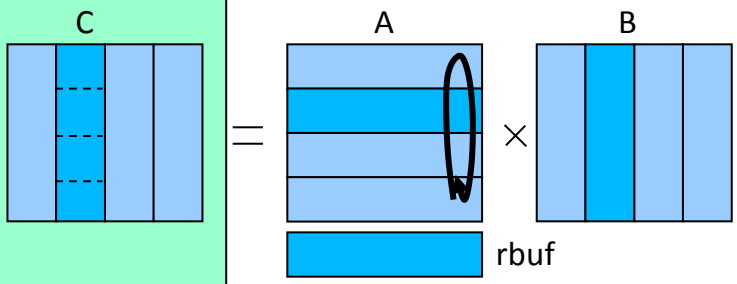

```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j, l=1;
   char tr = 't';
   for (i=0; i<n; i++) {
      #pragma omp task
      dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &m, b, &n, &beta, c, &n);
      c+=n;
      a+=m;
      b+=n;
   }

}
```

```
void callSendRecv(int m, int n,
                  double (*a)[m], int down,
                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```

- Overlap computation in tasks and communication in master

MxM @ MPI + OmpSs: MPI calls taskified

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {

   mxm( m, n, a, B, (double (*)[n])&C[i][0]);
   #pragma omp task in([n][m]a) out([n][m]rbuf)
   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;
   #pragma omp taskwait
}


free (orig_rbuf);
```



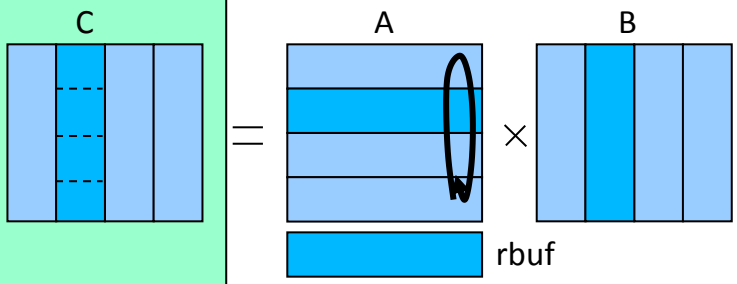

```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j, l=1;
   char tr = 't';
   for (i=0; i<n; i++) {
      #pragma omp task
      dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &m, b, &n, &beta, c, &n);
      c+=n;
      a+=m;
      b+=n;
   }

}
```

```
void callSendRecv(int m, int n,
                  double (*a)[m], int down,
                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```

- Overlap computation in tasks and communication in task

MxM @ MPI + OmpSs: MPI calls taskified

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {
   #pragma omp task in( [n][m]a, B ) inout ( C[i;n][0;n] )
   mxm( m, n, a, B, (double (*)[n])&C[i][0]);
   #pragma omp task in([n][m]a) out([n][m]rbuf)
   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;
   #pragma omp taskwait
}


free (orig_rbuf);
```



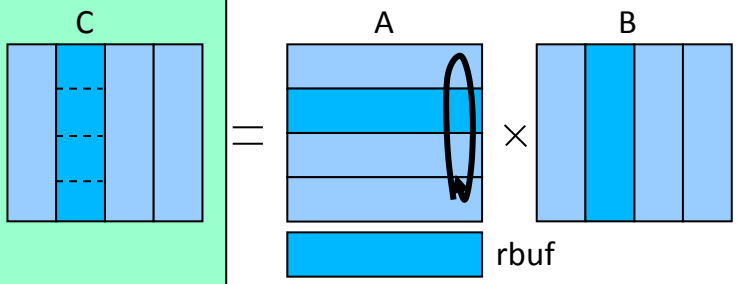

```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j, l=1;
   char tr = 't';
   for (i=0; i<n; i++) {
      #pragma omp task
      dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &m, b, &n, &beta, c, &n);
      c+=n;
      a+=m;
      b+=n;
   }
   #pragma omp taskwait
}
```

```
void callSendRecv(int m, int n,
                  double (*a)[m], int down,
                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```

- Nested
- Overlap between computation and communication in tasks

MxM @ MPI + OmpSs: MPI calls taskified

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {
   #pragma omp task in( [n][m]a, B ) inout ( C[i;n][0;n] )
   mxm( m, n, a, B, (double (*)[n])&C[i][0]);
   #pragma omp task in([n][m]a) out([n][m]rbuf)
   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;

}
#pragma omp taskwait

free (orig_rbuf);
```



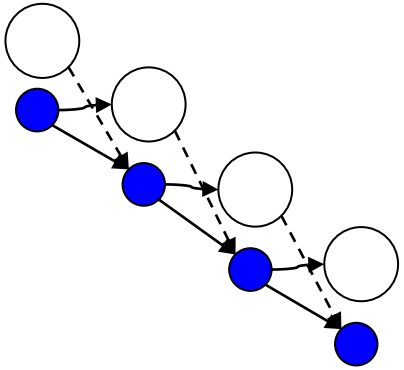

```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j, l=1;
   char tr = 't';
   for (i=0; i<n; i++) {
      #pragma omp task
      dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &m, b, &n, &beta, c, &n);
      c+=n;
      a+=m;
      b+=n;
   }
   #pragma omp taskwait
}
```

```
void callSendRecv(int m, int n,
                  double (*a)[m], int down,
                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```

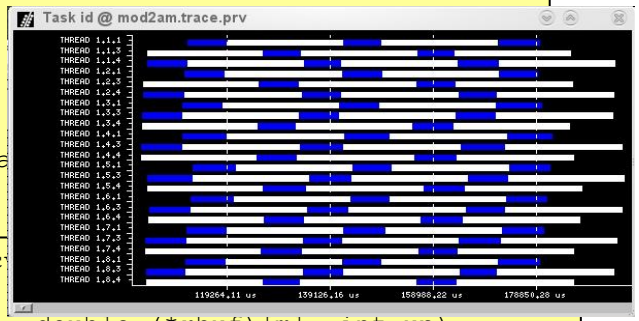
- Can start computation of next block of C as soon as communication terminates

MxM @ MPI + OmpSs: MPI calls taskified

```
// m,n: matrix size; local row/columns

double A[n][m], B[m][n], C[m][n], *a, *rbuf;

orig_rbuf = rbuf = (double (*)[m])malloc(m*n*sizeof(double));
if (nodes >1) { up=me<nodes-1 ? me+1:0; down=me>0 ? me-1:nodes-1;}
else { up = down = MPI_PROC_NULL; }

a=A;
i = n*me;  // first C block (different for each process)

for( it=0; it<nodes; it++ ) {
   #pragma omp task in( [n][m]a, B ) inout ( C[i;n][0;n] ) label (white)
   mxm( m, n, a, B, (double (*)[n])&C[i][0]);
   #pragma omp task in([n][m]a) out([n][m]rbuf) label (blue)
   callSendRecv(m, n, a, down, rbuf, up);

   //next C block circular
   i = (i+n)%m;
   //swap pointers
   ptmp=a; a=rbuf; rbuf=ptmp;

}
#pragma omp taskwait

free (orig_rbuf);
```



```
void mxm ( int m, int n, double *a, double *b, double *c ) {
   double alpha=1.0, beta=1.0;
   int i, j, l=1;
   char tr = 't';
   for (i=0; i<n; i++) {

      dgemm(&tr, &tr, &m, &n, &l, &alpha, a
      c+=n;
      a+=m;
      b+=n;
   }

}
```




```
void callSendRec
                                  double (*rbuf)[m], int up)
{
   int tag = 1000;
   int size = m*n;
   MPI_Status stats;

   MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag,
                 rbuf, size, MPI_DOUBLE, up, tag,
                 MPI_COMM_WORLD, &stats);
}
```

- Can obtain parallelism even without parallelizing the computation