Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Programming with OmpSs

Overlapping computation with communication using tasks

Computer Sciences Research Dept.
BSC

Agenda

- Motivation for a hybrid model
- Advantages of MPI/OmpSs
- Overlapping communication with computation
- Examples

- Contact: pm-tools@bsc.es
- Source code available from http://pm.bsc.es/ompss/

MPI + StarSs hybrid programming

- Why?
  - 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 exploits intra-node parallelism while StarSs can exploit node parallelism.
  - Overlap of communication and computation through the inclusion of communication in the task graph

Interaction between MPI and StarSs models

- 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
  - 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

Taskifying communications

- Communication Task:
  - Single MPI communication
  - Burst of MPI communications (+some small but potentially critical computation between them)

Taskifying communications

- Issues to consider
  - Possibly several concurrent MPI calls → thread safe MPI
    - Might not be available or really concurrent in some installations
  - Reordering of MPI calls + blocking calls + limited number of cores → potential to introduce deadlock
    - → need to control order of communication tasks
  - MPI task waste cores (busy waiting if communication partner delays or long transfer times)
    - Less problem as more and more cores per node become available

Deadlock potential when taskifying communications

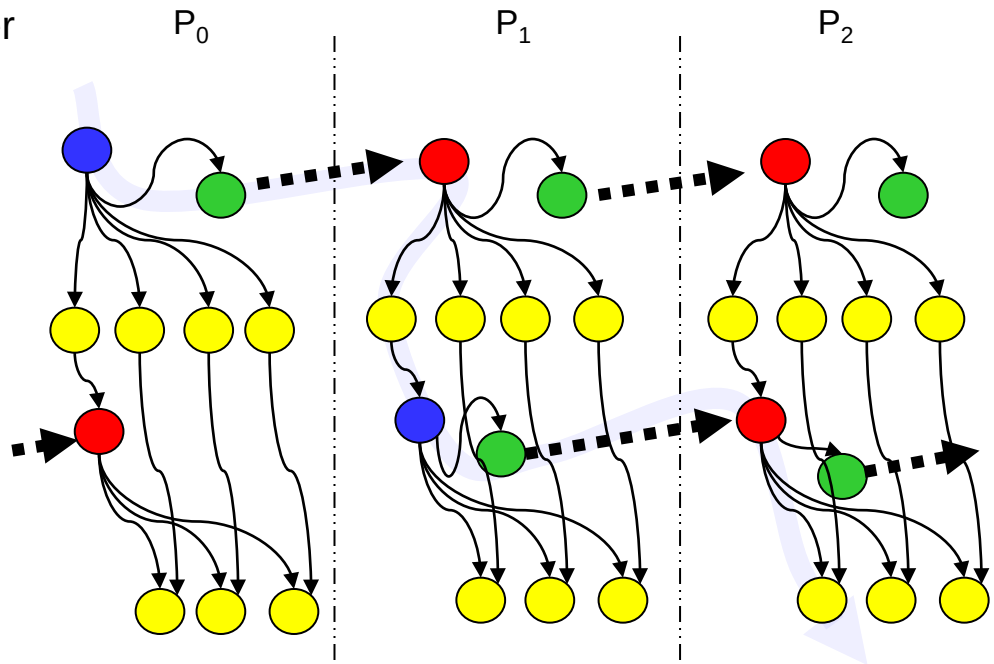
- Approaches to handle
  - Algorithmic structure may not expose loop and thus be deadlock safe
  - Enforce in order execution of communicating tasks
    - In the source code, possibly using a single sentinel specified as inout for all communication tasks
    - Specialized logical device
      - #target device (COMM_THREAD)
      - Modification in kernel scheduler to execute tasks allcoated to it in instantiation order
    - Serialization of communication can have an impact on performance
  - Virtualize communication resource. Allow infinite communication tasks
    - Integrated implementation of MPI interface in the OmpSs runtime
    - Implement blocking MPI calls by issuing nonblocking MPI calls blocking the task in NANOS and reactivate then when communication completes

Interaction between MPI and StarSs models

- Potential benefits
  - Gains deriving form overlap
    - Performance, Tolerance to noise, tolerance to low bandwidth
  - Load balance
    - Automatic within a node
  - Fighting Amdahl's Law
    - A hope for hybrid programming, a chance for lazzy programmers
  - Programmability simple alternative to nonblocking calls (point to point, collectives)

Hybrid MPI/StarSs

- Overlap communication/computation
- Extend asynchronous data-flow execution to outer level
- Linpack example: Automatic lookahead




```
…
for (k=0; k<N; k++) {
   if (mine) {
      Factor_panel(A[k]);
      send (A[k])
   } else {
      receive (A[k]);
      if (necessary) resend (A[k]);
   }
   for (j=k+1; j<N; j++)
      update (A[k], A[j]);
…
```

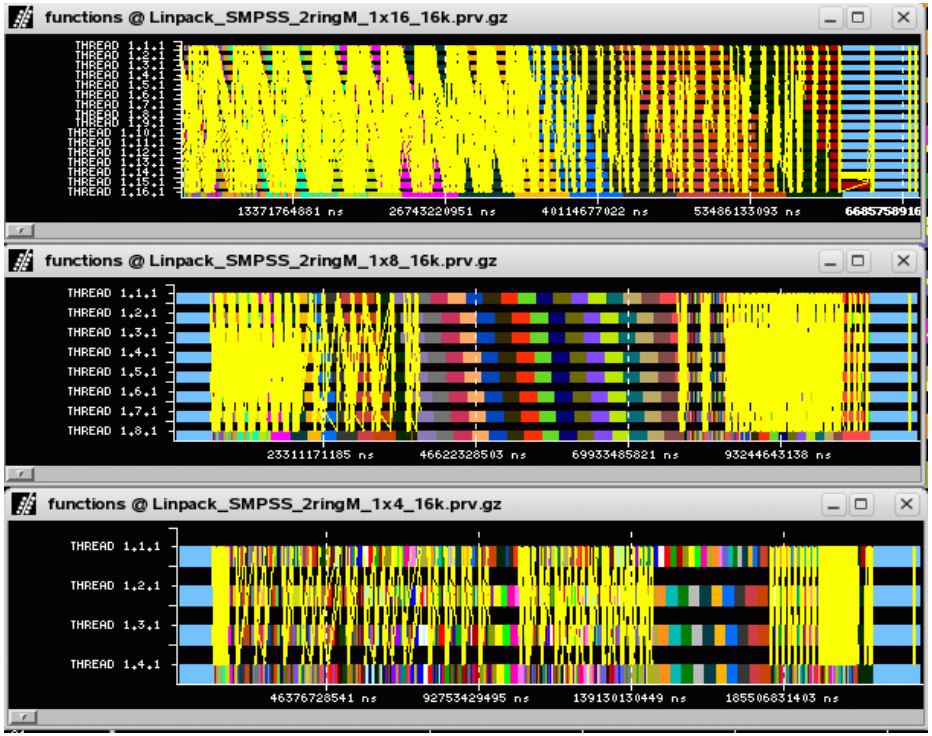
```
#pragma omp task inout([SIZE] A)
void Factor_panel(float *A);
#pragma omp task input([SIZE]A) inout([SIZE]B)
void update(float *A, float *B);
```

```
#pragma omp task input([SIZE]A)
void send(float *A);
#pragma omp task output([SIZE]A)
void receive(float *A);
#pragma omp task input([SIZE]A)
void resend(float *A);
```

V. Marjanovic, et al, "Overlapping Communication and Computation by using a Hybrid MPI/SMPSs Approach" ICS 2010

Hybrid MPI/OmpSs

- Overlap communication/computation
- Extend asynchronous data-flow execution to outer level
- Linpack example: Automatic lookahead



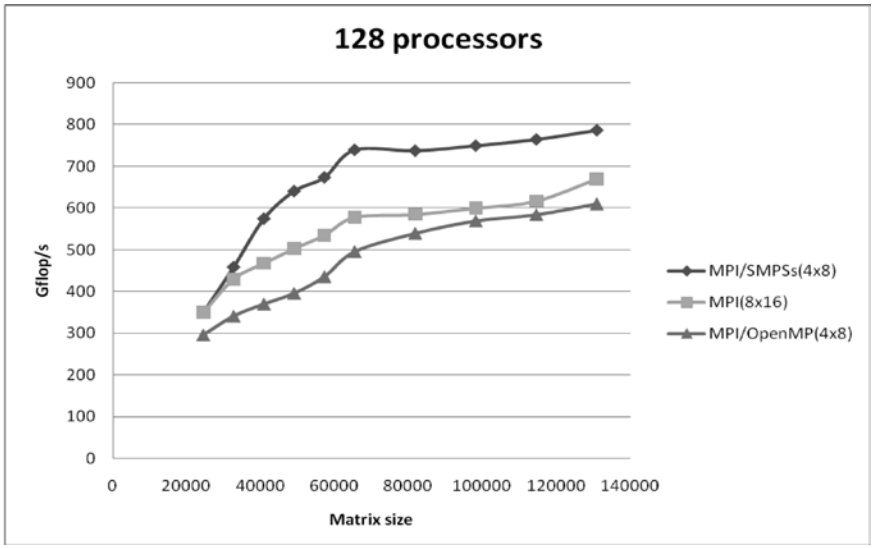

```
…
for (k=0; k<N; k++) {
   if (mine) {
      Factor_panel(A[k]);
      send (A[k])
   } else {
      receive (A[k]);
      if (necessary) resend (A[k]);
   }
   for (j=k+1; j<N; j++)
      update (A[k], A[j]);
…
```

```
#pragma omp task inout([SIZE]A)
void Factor_panel(float *A);
#pragma omp task input([SIZE]A) inout([SIZE]B)
void update(float *A, float *B);
```

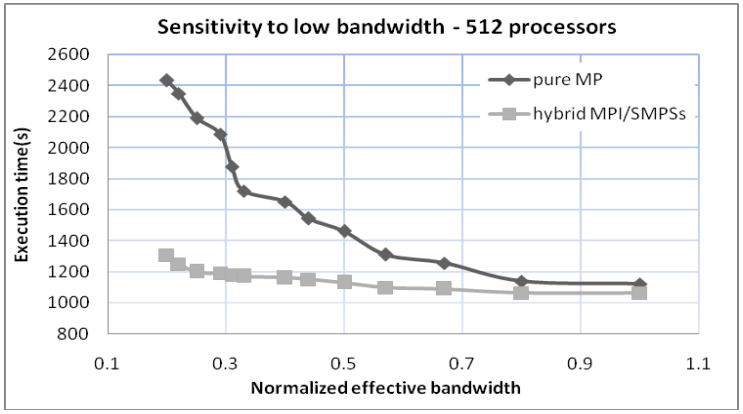
```
#pragma omp task input([SIZE]A)
void send(float *A);
#pragma omp task output([SIZE]A)
void receive(float *A);
#pragma omp task input([SIZE]A)
void resend(float *A);
```

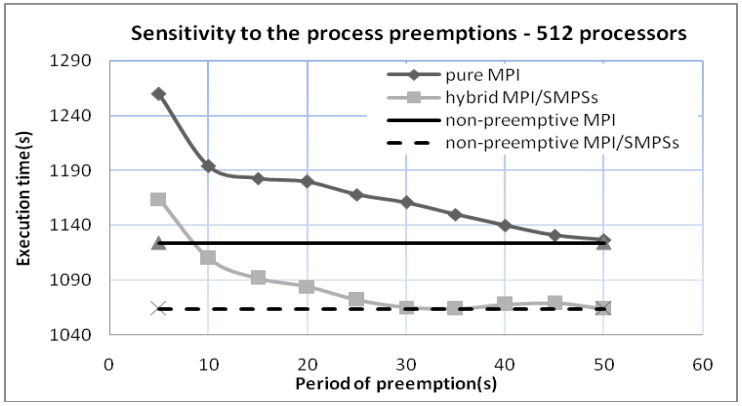
Hybrid MPI/OmpSs

- Performance
  - Higher at smaller problem sizes
  - Improved Load balance (less processes)
  - Higher IPC
  - Overlap communication/computation



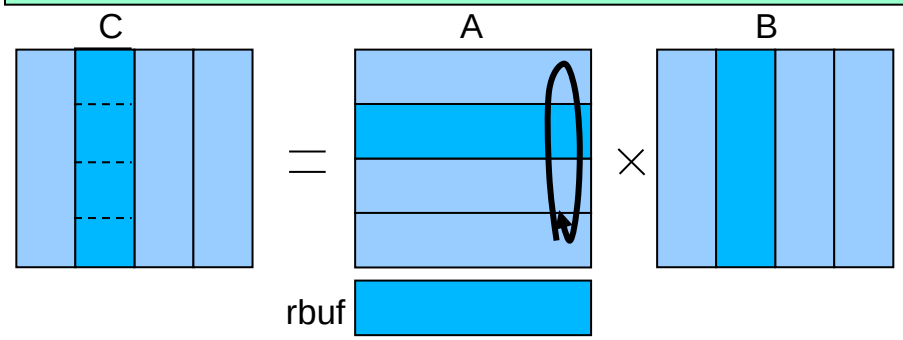
- Tolerance to bandwidth and OS noise

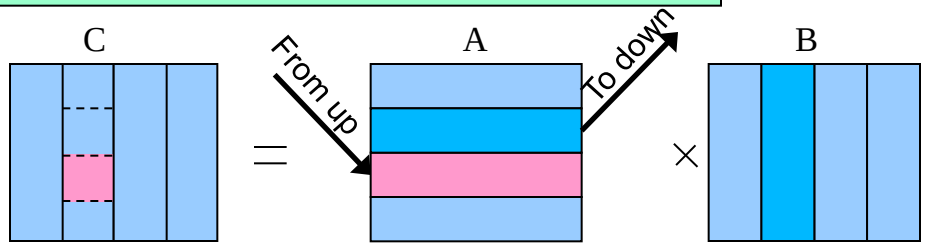




mxm: Original MPI code

```
for( i = 0; i < processes; i++ )
{
  stag = i + 1; rtag = stag;
  indx = (me + nodes - i )%processes;
  shift = ((offset[indx][0]/BS)*nDIM*BS*BS);
  size = vsize*l;
  if(i%2 == 0) {
    mxm ( lda, sizes[indx][0], lda, hsize, a, b, (c+shift) );
    MPI_Sendrecv (a, size, MPI_DOUBLE, down, stag, rbuf, size, MPI_DOUBLE, up, rtag, comm, &stats );
  } else {
    mxm (lda, sizes[indx][0], lda, hsize, rbuf, b, (c+shift) );
    MPI_Sendrecv (rbuf, size, MPI_DOUBLE, down, stag, a, size, MPI_DOUBLE, up, rtag, comm, &stats );
  }
}
```

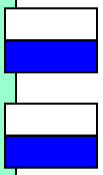


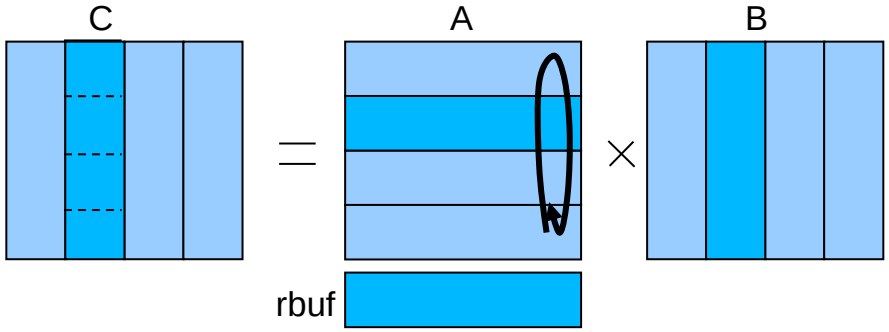


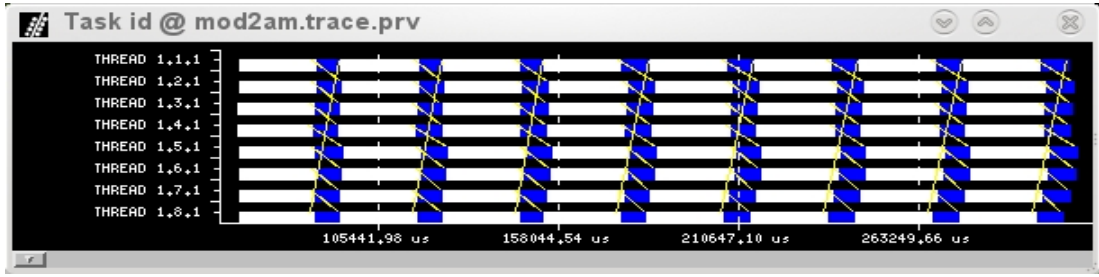

```
void mxm ( int lda, int m, int l, int n, double *a, double *b,
          double *c )
{
            double alpha=1.0, beta=1.0;
            int i, j;
            char tr = 't';
            dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &lda, b, &m, &beta, c, &m);
}
```

mxm: Original MPI code

```
for( i = 0; i < processes; i++ )
{
  stag = i + 1; rtag = stag;
  indx = (me + nodes - i )%processes;
  shift = ((offset[indx][0]/BS)*nDIM*BS*BS);
  size = vsize*l;
  if(i%2 == 0) {
    mxm ( lda, sizes[indx][0], lda, hsize, a, b, (c+shift) );
    MPI_Sendrecv (a, size, MPI_DOUBLE, down, stag, rbuf, size, MPI_DOUBLE, up, rtag, comm, &stats );
  } else {
    mxm (lda, sizes[indx][0], lda, hsize, rbuf, b, (c+shift) );
    MPI_Sendrecv (rbuf, size, MPI_DOUBLE, down, stag, a, size, MPI_DOUBLE, up, rtag, comm, &stats );
  }
}
```



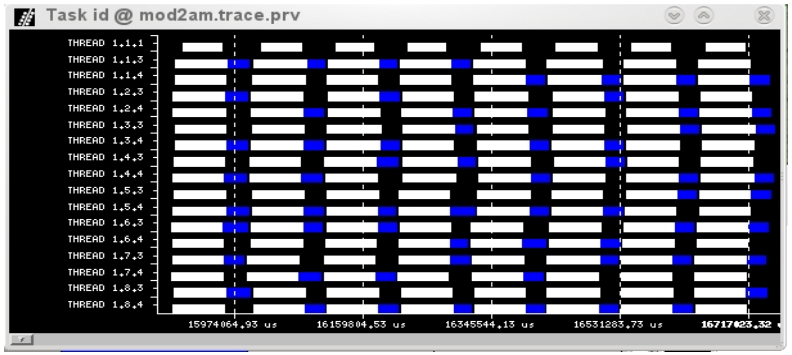





```
void mxm ( int lda, int m, int l, int n, double *a, double *b,
          double *c )
{
            double alpha=1.0, beta=1.0;
            int i, j;
            char tr = 't';
            dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &lda, b, &m, &beta, c, &m);
}
```

MPI+StarSs example – Matrix multiply

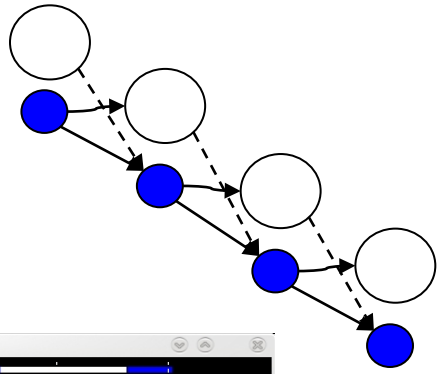
- Typical hybrid paralelization:
  - Parallelize computation phase.
  - Serialize communication part.

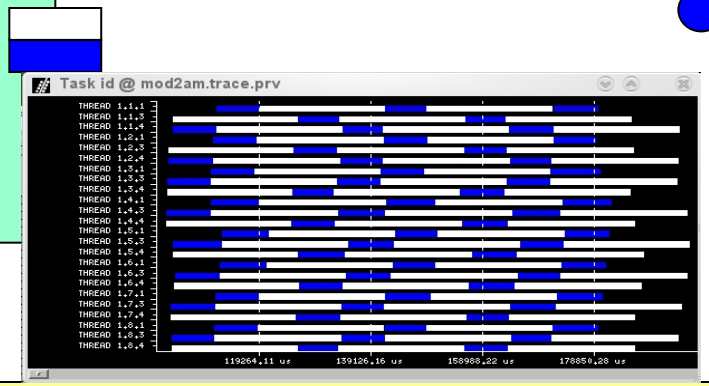


- How to overlap of communication and computation?
  - Using double buffering and
  - Asynchronous MPI calls
- Easier with StarSs...

Mod2am: Overlap communication and computation with StarSs

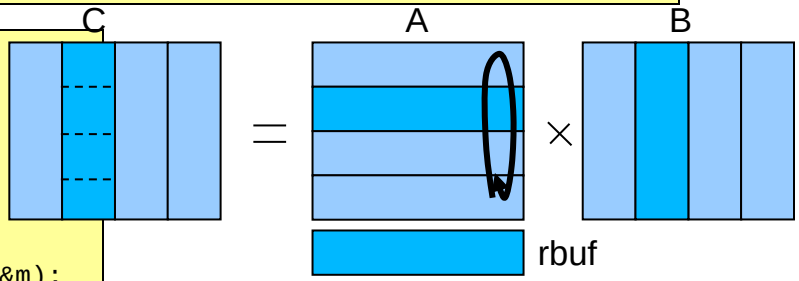
```
for( i = 0; i < processes; i++ )
{
  stag = i + 1; rtag = stag;
  indx = (me + nodes - i )%processes;
  shift = ((offset[indx][0]/BS)*nDIM*BS*BS);
  size = vsize*l;
  if(i%2 == 0) {
    mxm( lda, sizes[indx][0], lda, hsize, a, b, (c+shift) );
    callSendRecv (a, size, down, stag, rbuf, up, rtag);
  } else {
    mxm (lda, sizes[indx][0], lda, hsize, rbuf, b, (c+shift) );
    callSendRecv (rbuf, size, down, stag, a, up, rtag);
  }
}
```





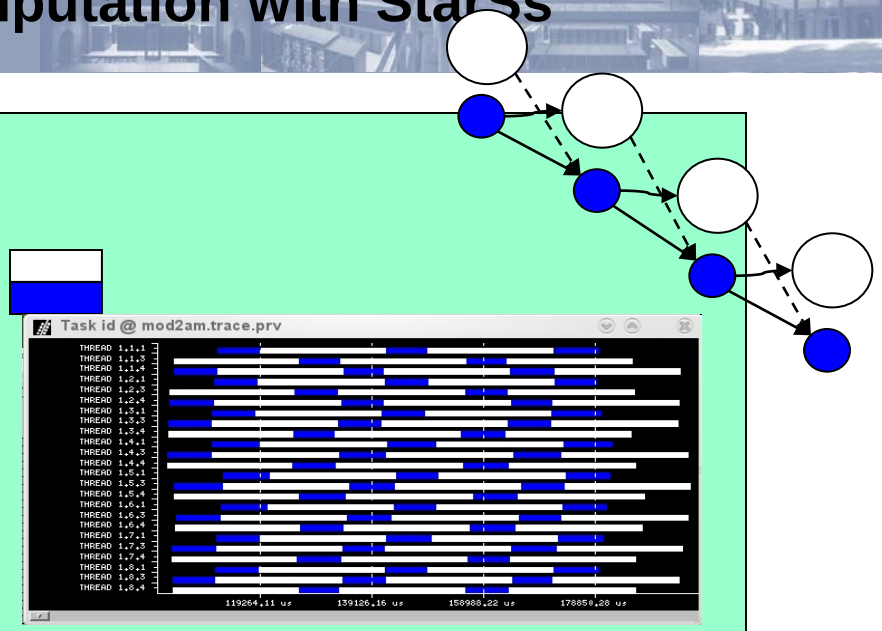

```
#pragma omp task input ([size]a) output ([size]rbuf)
void callSendRecv (double *a, int size, int down, int stag, double *rbuf, int up, int rtag)
{
  MPI_Status stats;
  MPI_Sendrecv( a, size, MPI_DOUBLE, down, stag, rbuf, size, MPI_DOUBLE, up, rtag, MPI_COMM_WORLD, &stats );
}
```

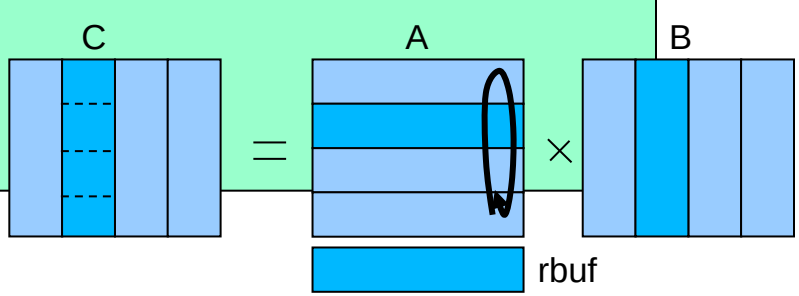
```
#pragma omp task input([m*l]a, [l*n]b) inout([m*n]c)
void mxm ( int lda, int m, int l, int n, double *a, double *b,
          double *c )
{
            double alpha=1.0, beta=1.0;
            int i, j;
            char tr = 't';
            dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &lda, b, &m, &beta, c, &m);
}
```



mxm: Overlap communication and computation with StarSs

```
    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)
    size = m*n;

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

      #pragma omp task input (a[0:n-1], B[0:m-1])\
             inout (C[i:i+n-1][0:n-1]) firstprivate (n,m)
      cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
            n, n, m, 1.0, (double *)a, m, (double *)B, n, 1.0, (double *)&C[i][0], n);

      if (it < nodes-1) {
         #pragma omp task input (a[0:n-1]) output (rbuf[0:n-1]) inout(stats) firstprivate (size,m,n,tag,down,up)
         MPI_Sendrecv( a, size, MPI_DOUBLE, down, tag, rbuf, size, MPI_DOUBLE, up, tag, MPI_COMM_WORLD, &stats );
      }

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

#pragma omp taskwait
    free (orig_rbuf);
```





mxm: Overlap communication and computation with nesting

```
 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);

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

#pragma omp taskwait
    free (orig_rbuf);
```

```
#pragma omp task input(A[0:n-1], B[0:m-1]) inout(C[0:m-1])
void mxm (int m, int n, double (*A)[m], double (*B)[n], double (*C)[n])
{
   if ((m <= n) ) {
        dgemm_cblas(m, n, A, B, C);
   } else {
      int j, J;
      int nb=m/n;

      double (*Abp)[n];   // temporal block for contiguous A
      Abp = (double (*)[n]) malloc (n*n*sizeof(double));

      for(J = 0; J < nb; J++) {
            gather_block(m, A, 0, J, n, Abp);
            //get comtiguous temporal copy of block of A
            mxm ( n, n, Abp, (double (*)[n])&B[J*n][0], C);
            //contribute to block [n][n] of C
      }

#pragma omp taskwait
      free (Abp);
   }
}
```

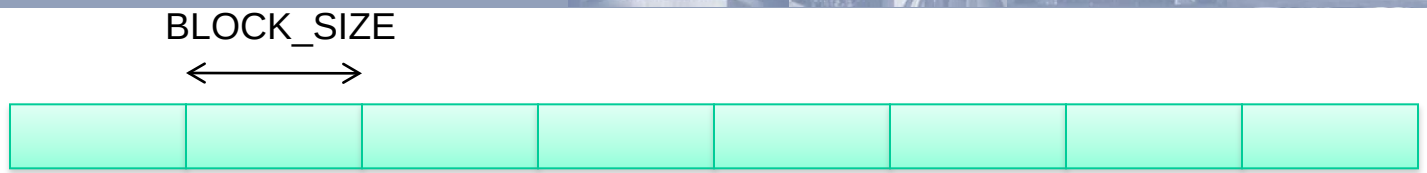
mxm: Overlap communication and computation with nesting

```
 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 (differen

    for( it = 0; it < nodes; it++ ) {
       mxm( m, n, a, B, (double (*)[n])&C
       callSendRecv(m, n, a, down, rbuf, 

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

#pragma omp taskwait
    free (orig_rbuf);
```

```
#pragma omp task input (a[0:n-1]) output (rbuf[0:n-1])
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 );
}
```

```
      int j, J;
      int nb=m/n;

      double (*Abp)[n];   // temporal block for contiguous A
      Abp = (double (*)[n]) malloc (n*n*sizeof(double));

      for(J = 0; J < nb; J++) {
            gather_block(m, A, 0, J, n, Abp);
            //get comtiguous temporal copy of block of A
            mxm ( n, n, Abp, (double (*)[n])&B[J*n][0], C);
            //contribute to block [n][n] of C
      }

#pragma omp taskwait
      free (Abp);
   }
}
```

Pingpong

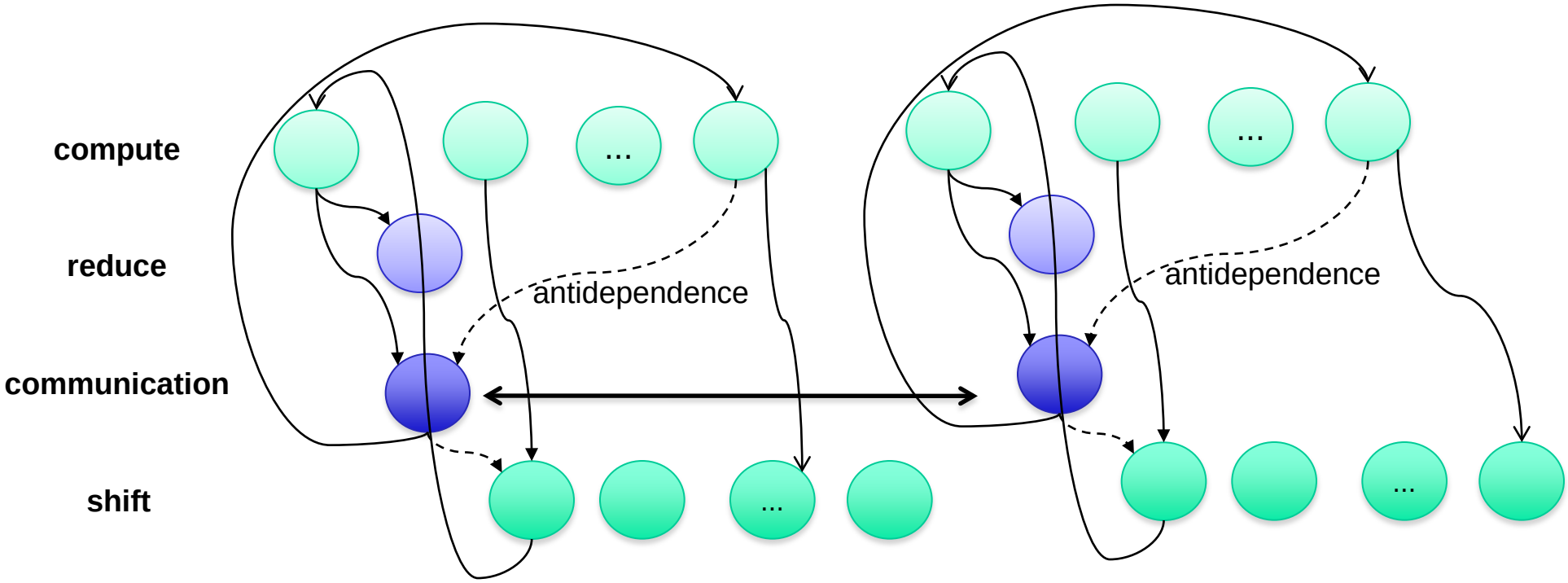



```
#pragma omp task inout([n]local_array)
void compute(double *local_array, int n);

#pragma omp task input([n]local_array) inout([1]sum)
void reduce(double *local_array, double *sum, int n);

#pragma omp task input([n]bufsend) output([n]bufrecv)
void communication(int partner, double *bufsend, double *bufrecv, int n);

#pragma omp task input([n]array_right) output([n]array_left)
void shift(double *array_right, double *array_left, int n);
```

```
for(i = 0; i < NUM_OF_ITE; i++){

     for(j = 0; j < SIZE_OF_ARRAY; j+=BLOCK_SIZE)
       compute(&array[j], BLOCK_SIZE);
     reduce(&array[0], &sum, BLOCK_SIZE);
     communication (partner, &array[0], &array[SIZE_OF_ARRAY-BLOCK_SIZE], BLOCK_SIZE);

     for(j = 0; j < SIZE_OF_ARRAY-BLOCK_SIZE; j+=BLOCK_SIZE)
       shift(&array[j+BLOCK_SIZE], &array[j], BLOCK_SIZE);

}
```

Task Dependence Graph

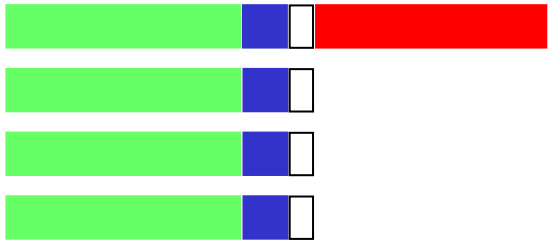
- Each process executes the same code
  - Not all dependences shown



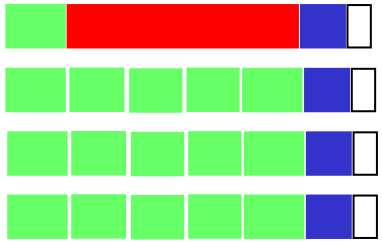
Fighting Amdahl's law: StarSs. a chance for lazy programmers

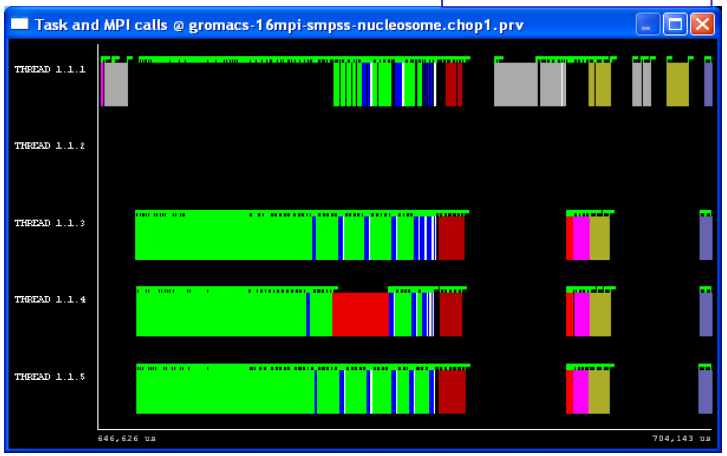
Four loops/routines
Sequential program order

OpenMP
not parallelizing one loop



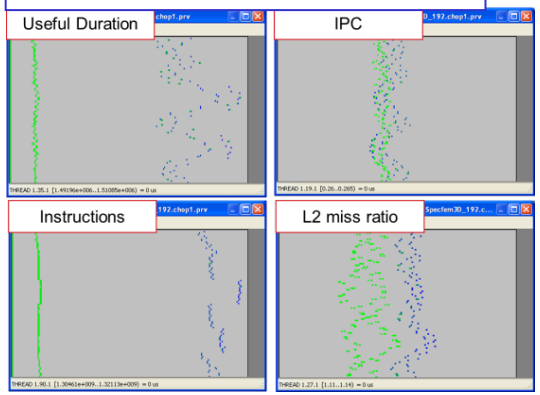
SMPSs
not parallelizing one loop



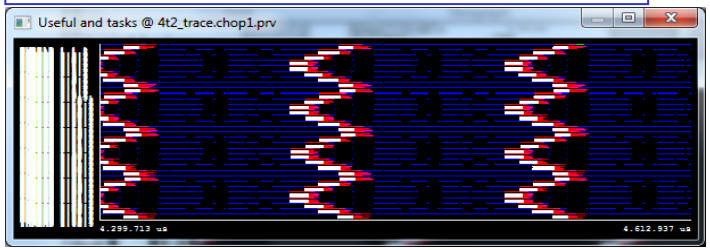


Load imbalance is everywhere

Old SPECEFEM3D version

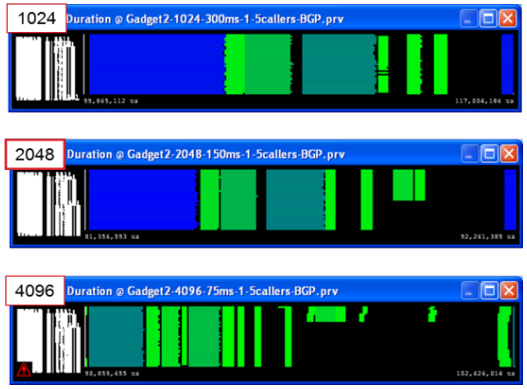


PSC @ 32x4 cores MPI/SMPSs

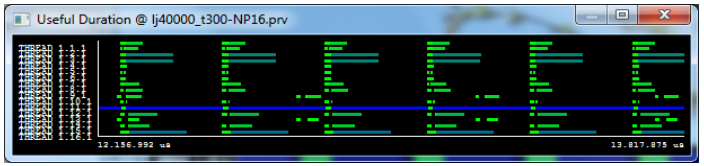


Gadget @ 1024 - 4096 cores

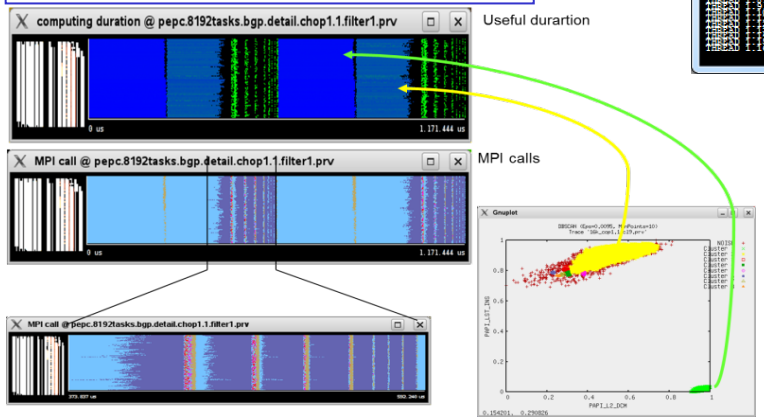
Useful duration one iteration (not same scale!!)



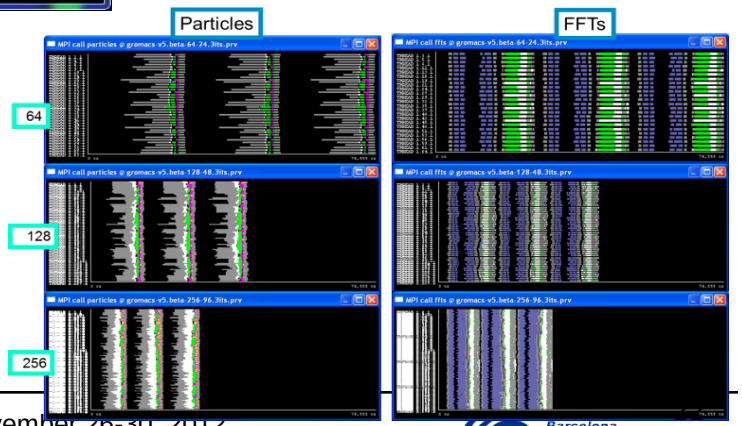
LS1 @ 16 cores



Original PEPC @ 8K cores

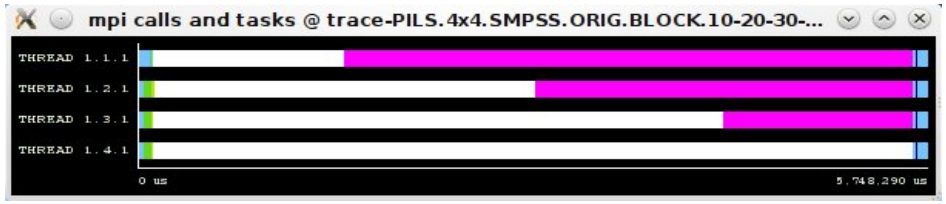


Gromacs @ 64-265 cores



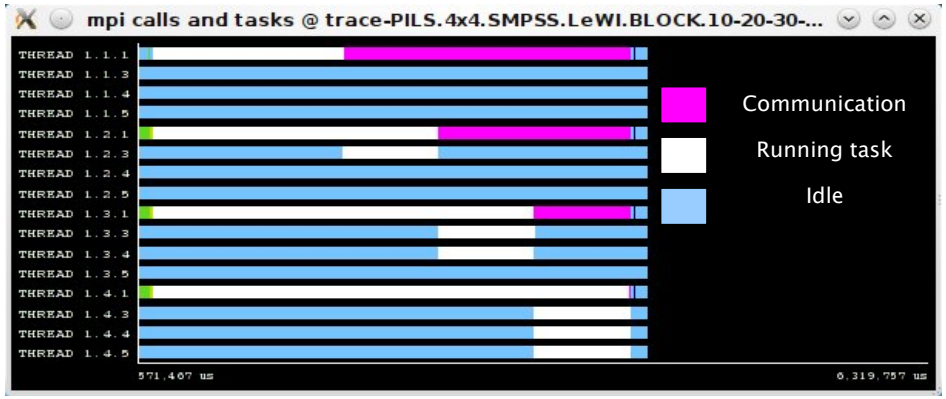
Dynamic Load Balancing: LeWI

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



4 MPI processes @ 4 cores node

- LeWI: Lend core When Idle



- Overcome Amdahl's law in hybrid programming
  - Enables partial node level parallelization
  - hope for lazy programmers … and us all

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

MPI + StarSs

- **Hybrid MPI + OmpSs: overcome weaknesses, leverage virtues**
  - **Performance**: Overcomes the too synchronous structure of MPI/OpenMP, propagating the dataflow asynchronous behavior to the MPI level.
  - **Flexibility**: making the application malleable
    - Introduces the possibility of flexible resource management in otherwise rigid process structure of MPI or MPI/OpenMP
    - Supports automatic fine grain load balance, reaction to faults and dynamic system level resource management policies.
  - **Productivity**: Allows for very flexible overlap with simple code structures
  - **Portability**: Offers incremental parallelization even for heterogeneous systems