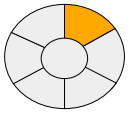
Agenda

- Hybrid model MPI/OmpSs
- MPI/OmpSs handson

- Slides and MPI/OmpSs source code: /tmp/tutorial_ompss_PATC_MPI_OmpSs.ppt
- /tmp/tutorial_PATC_MPI_OmpSs.tar.gz
- Contact: pm-tools@bsc.es
- Source code available from http://pm.bsc.es/ompss/

Hybrid MPI/OmpSs



MPI + StarSs hybrid programming

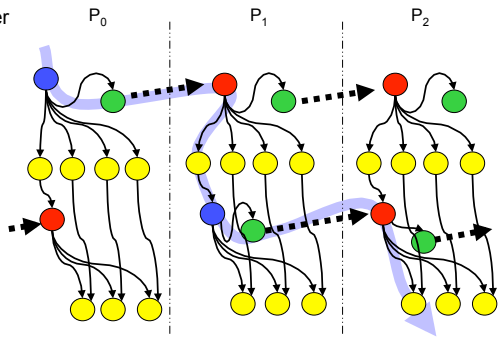
- Why?
  - MPI is here to stay.
  - A lot of HPC applications already written in MPI.
  - MPI scales well to tens/hundreds of thousands of nodes.
  - MPI exploits intra-node parallelism while StarSs can exploit node parallelism.
  - Overlap of communication and computation through the inclusion of communication in the task graph

MPI + StarSs

- **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

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]);
…
```

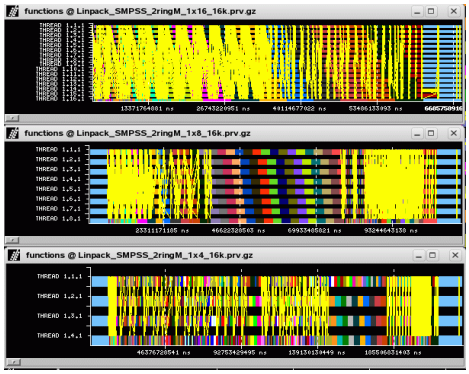
```
#pragma css task inout(A[SIZE])
void Factor_panel(float *A);
#pragma css task input(A[SIZE]) inout(B[SIZE])
void update(float *A, float *B);
```

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

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

Hybrid MPI/SMPSs

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



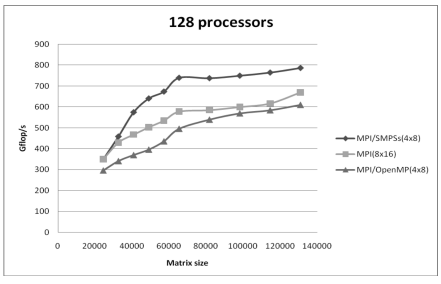

```
…
for (k=0; k<N; k++) {
   if (mine) {
      Factor_panel(A[k]);
      send (A[k])
   } else {
      receive (A[k]);
      if (necessary) resend (A[k]);
   }
   for (j=k+1; j<N; j++)
      update (A[k], A[j]);
…
```

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

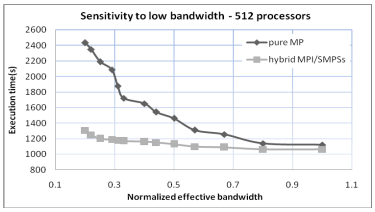
```
#pragma css task input(A[SIZE])
void send(float *A);
#pragma css task output(A[SIZE])
void receive(float *A);
#pragma css task input(A[SIZE])
void resend(float *A);
```

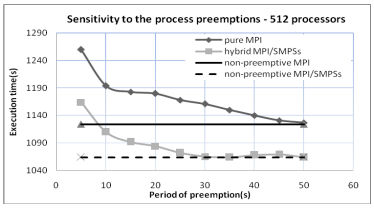

Hybrid MPI/SMPSs

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



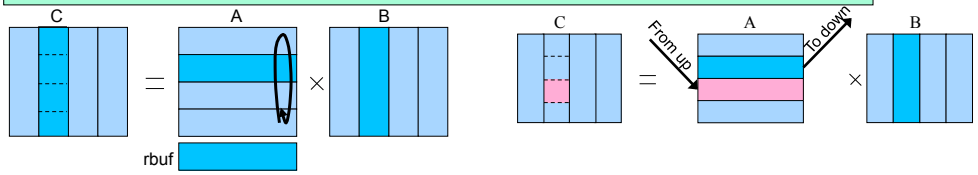
- Tolerance to bandwidth and OS noise





mxm: Original MPI code

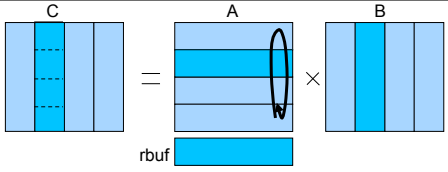
```
for( i = 0; i < processes; i++ )
{
  stag = i + 1; rtag = stag;
  indx = (me + nodes - i )%processes;
  shift = ((offset[indx][0]/BS)*nDIM*BS*BS);
  size = vsize*l;
  if(i%2 == 0) {
    mxm ( lda, sizes[indx][0], lda, hsize, a, b, (c+shift) );
    MPI_Sendrecv (a, size, MPI_DOUBLE, down, stag, rbuf,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

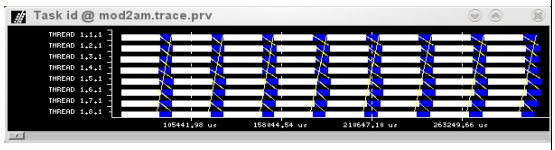



```
void mxm ( int lda, int m, int l, int n, double *a, double *b,
           double *c )
{
             double alpha=1.0, beta=1.0;
             int i, j;
             char tr = 't';
             dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &lda, b, &m, &beta, c, &m);
}
```


Mod2am: 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 );
  }
}
```

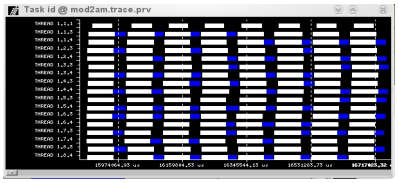





```
void mxm ( int lda, int m, int l, int n, double *a, double *b,
           double *c )
{
             double alpha=1.0, beta=1.0;
             int i, j;
             char tr = 't';
             dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &lda, b, &m, &beta, c, &m);
}
```

MPI+StarSs example – Matrix multiply

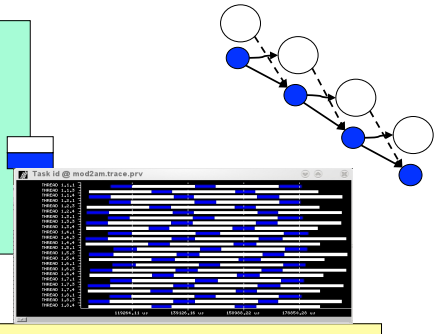
- Typical hybrid paralelization:
  - Parallelize computation phase.
  - Serialize communication part.



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

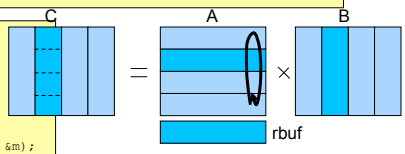
Mod2am: Overlap communication and computation with StarSs

```
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 );
}
```

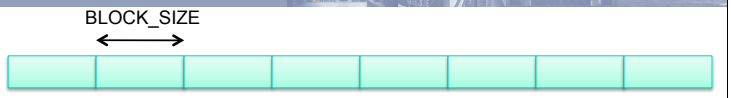
```
#pragma omp task input([m*l]a, [l*n]b) inout([m*n]c)
void mxm ( int lda, int m, int l, int n, double *a, double *b,
           double *c )
{
            double alpha=1.0, beta=1.0;
            int i, j;
            char tr = 't';
            dgemm(&tr, &tr, &m, &n, &l, &alpha, a, &lda, b, &m, &beta,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



Problems taskifying communications

- Problems:
  - Possibly several concurrent MPI calls
    - Need to use thread safe MPI
  - Reordering of MPI calls + limited number of cores → potential source of deadlocks
    - Need to control order of communication tasks
    - Use of sentinels if necessary
  - MPI task waste cores (busy waiting if communication partner delays or long transfer times)
    - An issue today, may be not with many core nodes.

Pingpong

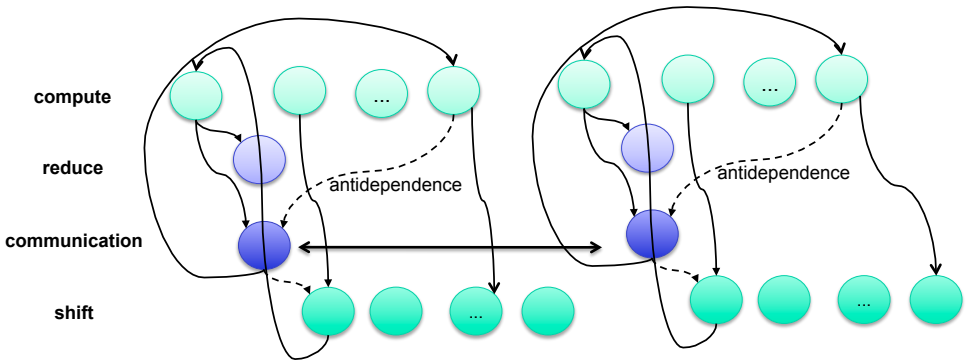



```
#pragma omp task inout([n]local_array)
void compute(double *local_array, int n);

#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



MPI/OmpSs environment

- Compiling and running
  - Use of regular mcc compiler (or imcc)
  - Link with MPI library
  - Use scripts to submit job to queues
- Generating traces
  - compile with --instrument
  - export NX_INSTRUMENTATION=extrae
  - Need to preload instrumentation library
    - We are instrumenting at the same time MPI and OmpSs
    - LD_PRELOAD="/gpfs/apps/NVIDIA/PM/extrae/lib/libnanosmpitrace.so:/opt/mpi/bullxmpi/1.1.11.1/lib/libmpi.so"

MPI/OmpSs hands-on

- Starting code: pingpong_ompss/pinpong_ompss.c
  - MPI code, with OmpSs pragmas
  - Compile using Makefile
  - Execute and generate trace with the my_job.sh script
    - mnsubmit my_job.sh
  - Analyse the generated tracefile
    - Regular cfgs
    - Specific cfgs in MPI_OmpSs directory
    - Specific cfgs for MPI in mpi directory

MPI/OmpSs hands-on

- Starting code: mxm-simple/matmul.c
  - MPI code, non completed OmpSs pragmas
  - Create first basic version that enables to overlap communication with computation
    - Complete the inlined compiler directives to the functions cblas_dgemm and MPI_Sendrecv
    - Compile using Makefile
    - Execute and generate trace with the my_job_mxm_s.sh script
      - msub my_job_mxm_s.sh
    - Analyse the generated tracefile
    - Execute and generate traces with different processors counts (MPI processes and OMP_NUM_THREADS)

MPI/OmpSs hands-on

- Starting code: mxm-nested/matmul.c
  - Complete required compiler directives to obtain a version with nested tasks
  - Compile
  - Execute and generate tracefile
  - Analyse the generated tracefile

MPI/OmpSs hands-on

- Starting code: nbody
  - Complete required compiler directives to obtain a version with MPI and OmpSs with CUDA tasks
  - Compile
  - Execute and generate tracefile
    - The job script compares the result
    - Check that the result is correct
    - If it is not correct -> think about missing taskwaits!!!
  - Analyse the generated tracefile

Conclusions

- Future programming models should:
  - Enable productivity and portability
  - Support for heterogeneous/hierarchical architectures
  - Support asynchrony → global synchronization in systems with large number of nodes is not an answer anymore
  - Be aware of data locality
- OmpSs is a proposal that enables:
  - Incremental parallelization from existing sequential codes
  - Data-flow execution model that naturally supports asynchrony
  - Nicely integrates heterogeneity and hierarchy
  - Support for locality scheduling
  - Active and open source project:

    http://pm.bsc.es/ompss