TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Tutorial on Hybrid MPI/OpenMP programming

Victor Eijkhout

SSiASC 2016

TACC

Justification

Clusters have a hybrid structure with distributed and shared memory component. This course teaches the concepts of hybrid computing and process/thread affinity.

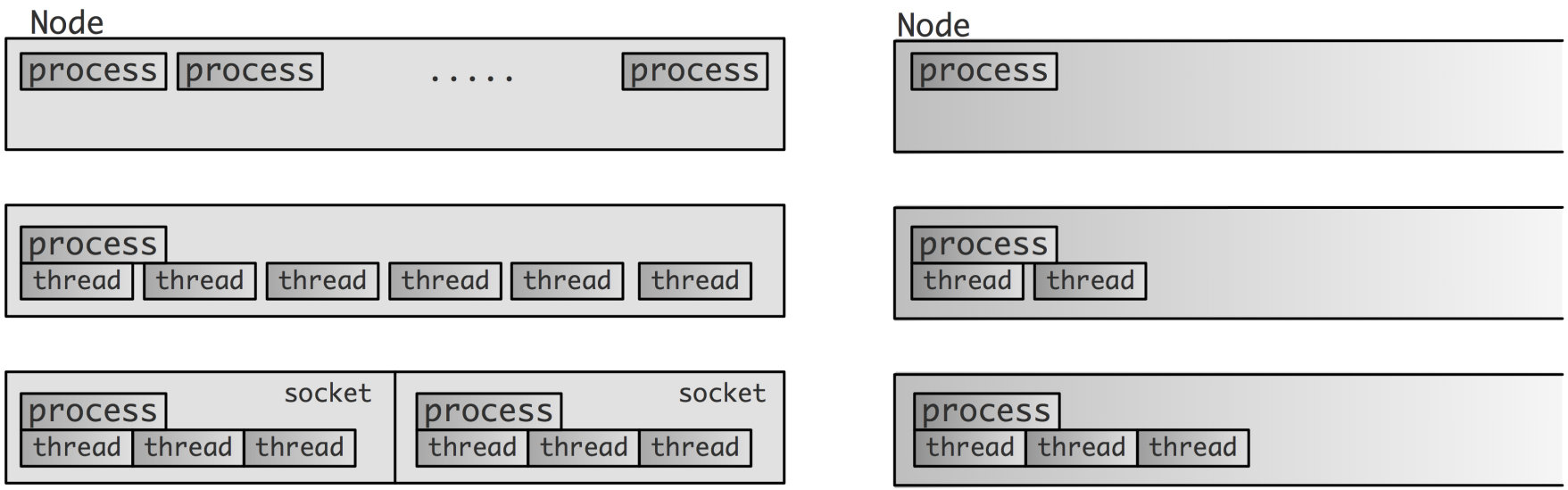
Hybrid computing

Cluster structure

- Cluster nodes connected through network
  (network topology?)
- Node has sockets: processor chips
  shared memory but slightly unequal access
- Socket has cores
  fairly symmetric, but watch out for KNL
- Core can have hardware/hyper threads
  (sometimes called virtual core vs physical core)

Parallelization strategies

- Pure MPI
- One MPI per node, full OpenMP on the node
- One MPI per socket, fewer OpenMP threads than total cores
- $\Rightarrow$ MPI process is a 'container' for OpenMP threads



How do you run this?

- Full MPI

```
#$ SBATCH -N 100
#$ SBATCH -n 1600
export OMP_NUM_THREADS=1
ibrun tacc_affinity yourprogram
```

- Maximal OpenMP

```
#$ SBATCH -N 100
#$ SBATCH -n 100
export OMP_NUM_THREADS=16
ibrun tacc_affinity yourprogram
```

- Process per socket:

```
#$ SBATCH -N 100
#$ SBATCH -n 200
export OMP_NUM_THREADS=8
ibrun tacc_affinity yourprogram
```

Benefits of hybrid computing

- No obvious speedup: all cores are active in all cases.
- Secondary Amdahl effect: watch our for code that is MPI parallel but not OpenMP parallel.
- MPI codes can run somewhat unsychronized; OpenMP has more barriers.
- On the other hand, OpenMP is easier to load balance.
- Only one big messsage between nodes, rather than many small in full MPI case.
- Fewer MPI processes means less buffer space.

tacc_affinity

`tacc_affinity` does two things:

- Unset affinity setting for `mvapich2`
- Set process placement to something reasonable.

Exercise 1 (`decomp`)

- Compile the `decomp.c` program.

```
mpicc -fopenmp decomp.c -o decomp
```

- Submit the `hybridscript`.
- Study the output. What conclusion do you draw?

Setting up MPI for hybrid computing

```
int MPI_Init_thread(int *argc, char ***argv,
    int required, int *provided)
```

`MPI_THREAD_SINGLE` Only a single thread will execute.

`MPI_THREAD_FUNNELLED` The program may use multiple threads, but only the main thread will make MPI calls.

`MPI_THREAD_SERIAL` The program may use multiple threads, all of which may make MPI calls, but there will never be simultaneous MPI calls in more than one thread.

`MPI_THREAD_MULTIPLE` Multiple threads may MPI calls, without restrictions.

Single

```
MPI_Init_thread(0,0,MPI_THREAD_SINGLE,&ipr);
MPI_Recv( ... );
// no OpenMP
```

Funnelled

```
MPI_Init_thread(0,0,MPI_THREAD_FUNNELLED,&ipr);
#pragma omp parallel
{
#pragma omp master
  MPI_Recv( ... );
#pragma omp barrier // probably
  ...
}
```

Serial

```
MPI_Init_thread(0,0,MPI_THREAD_SERIAL,&ipr);
#pragma omp parallel
{
#pragma omp critical
  MPI_Recv( ... );
  ...
}
```

Multiple

```
MPI_Init_thread(0,0,MPI_THREAD_MULTIPLE,&ipr);
#pragma omp parallel
{
  MPI_Recv( ... );
  ...
}
```

Finding shared memory in MPI

```
C:
int MPI_Comm_split_type(
  MPI_Comm comm, int split_type, int key,
  MPI_Info info, MPI_Comm *newcomm)

Fortran:
MPI_Comm_split_type(comm, split_type, key, info, newcomm, ierror)
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, INTENT(IN) :: split_type, key
TYPE(MPI_Info), INTENT(IN) :: info
TYPE(MPI_Comm), INTENT(OUT) :: newcomm
INTEGER, OPTIONAL, INTENT(OUT) :: ierror

Python:
MPI.Comm.Split_type(
  self, int split_type, int key=0, Info info=INFO_NULL)
```

How to read routine prototypes: **??**.

Process and thread affinity

Definition

Affinity is the mapping between

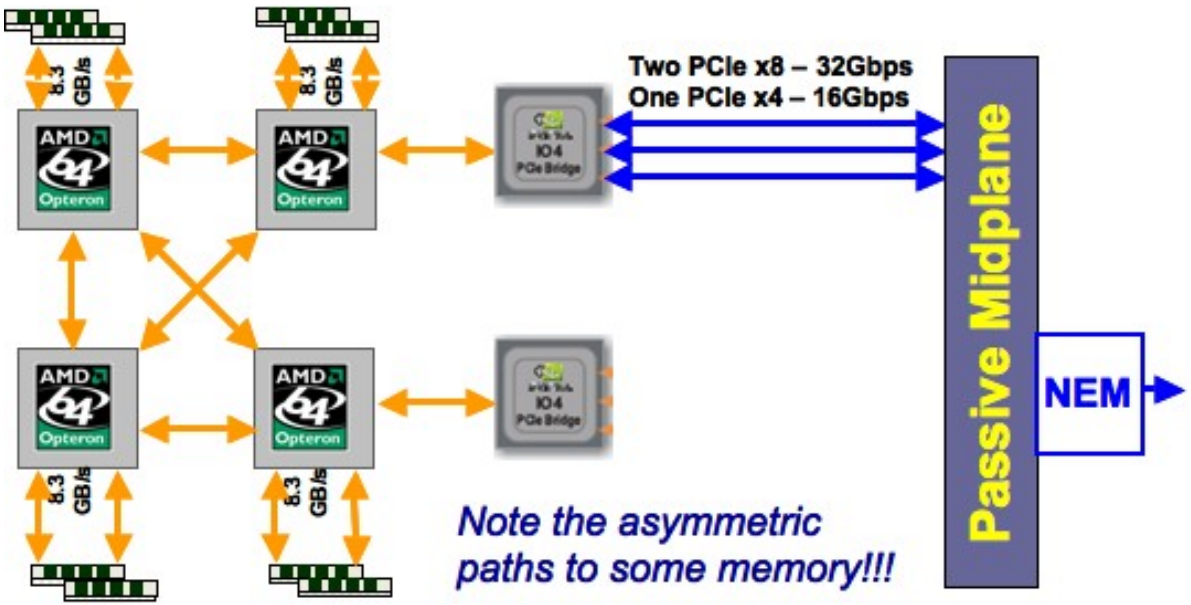
- software concepts such as processes and threads, and
- hardware concepts such as nodes, sockets, cores;

in particular as it concern the efficiency of

- memory transfers,
- communication and synchronization.

Examples

- In pure MPI mode, processes on the same node communicate faster than ones on diferent nodes.
- If the network has a topology, speed between nodes is not uniform.
- Communication between sockets on one node need not be uniform (see figure)
- Each socket has local memory: shared memory is not uniform.
- In try hybrid mode, it matters where you place processes and threads.
- The operating system is allowed to move processes and threads on a node. . .



NUMA

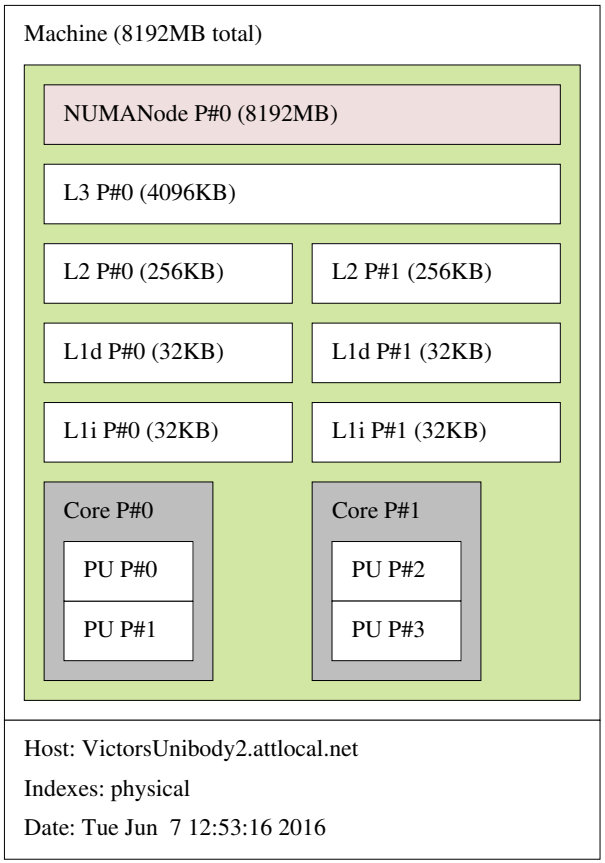
Non-Uniform Memory Access:
shared memory, but not every location same speed of access

- Sockets
- Caches in a socket
- ‘Distributed shared memory’; see SGI UV

Exploring the hardware

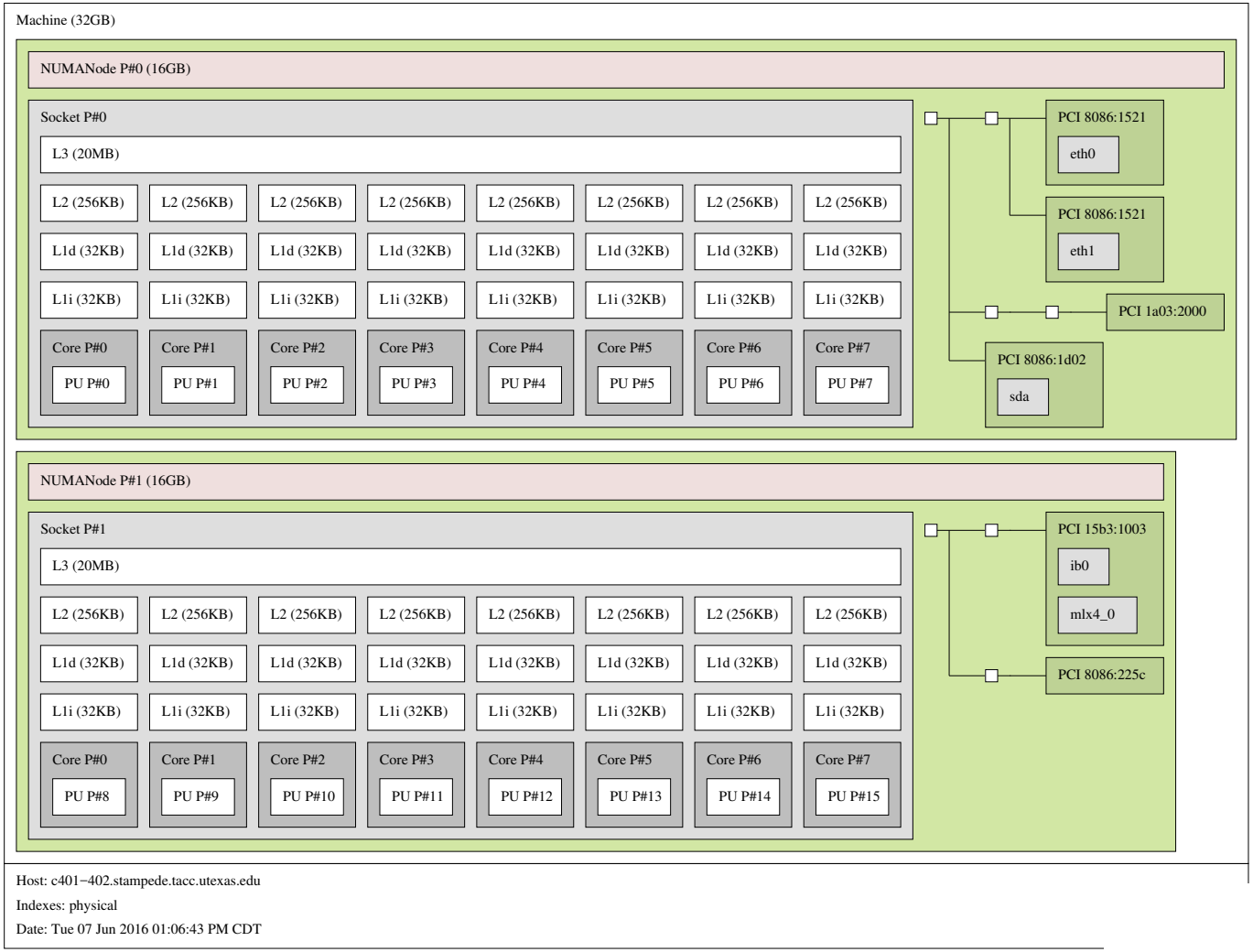
`hwloc` package: `lstopo`

Start with my laptop, an Intel i5:



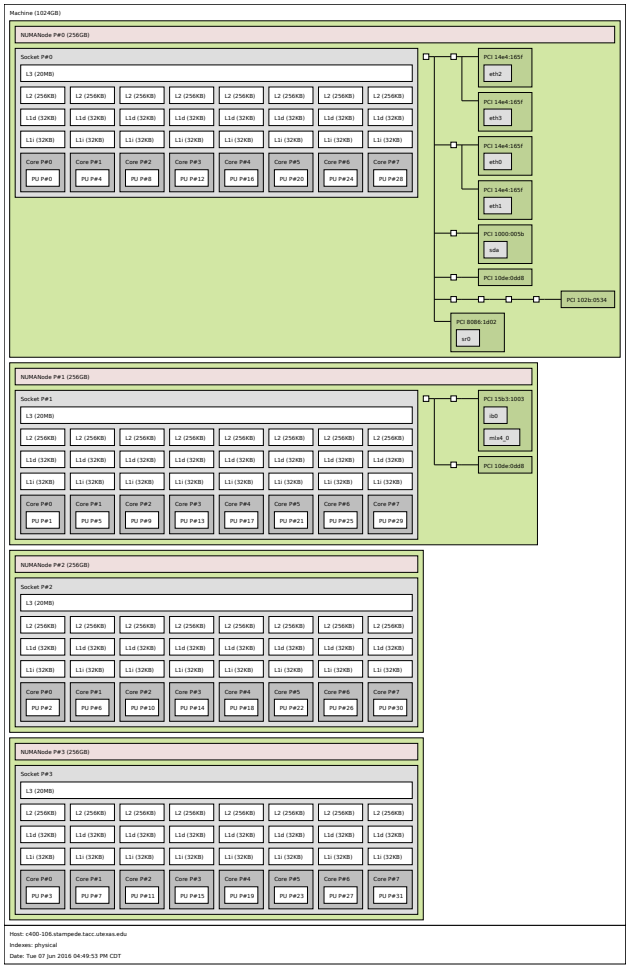
Stampede compute node

Two sockets; each eight cores



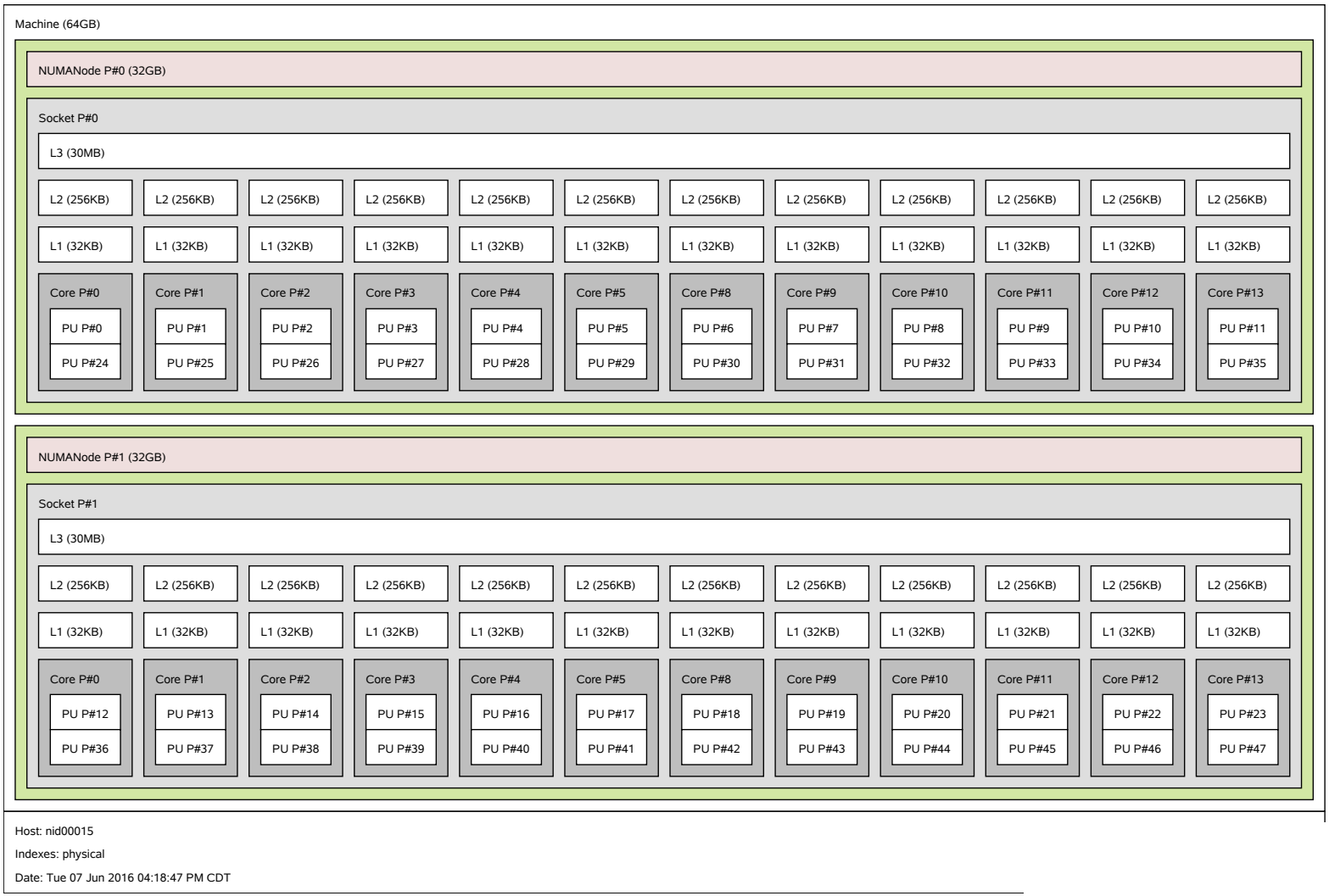
Stampede largemem node

Four sockets; each eight cores



Lonestar5 compute node

Two sockets; each twelve cores, two hyperthreads



Knight's Landing

One socket, four numa domains, 16 cores each, four hyperthread

Affinity control

- You can spell it out yourself: `numactl`, `OMP_PLACES` but that is quite tedious.
- Process affinity with `tacc_affinity`
  (wrapper around numactl)
- Set `OMP_PROC_BIND=true` to prevent thread migration. That is usually enough.

First touch

- Allocated memory only created when you write to it.
- $\Rightarrow$ If one core initializes all the memory, it becomes bound to that socket

Wrong:

```
double *x = (double*) malloc(N*sizeof(double));

for (i=0; i<N; i++)
  x[i] = 0;

#pragma omp parallel for
for (i=0; i<N; i++)
  .... something with x[i] ...
```

Repair: initialize in parallel, with same schedule.

Exercise 2 (`central`)

Finish the following fragment and run it with first all the cores of one socket, then all cores of both sockets. (If you know how to do explicit placement, you can also try fewer cores.)

```
  for (int i=0; i<nlocal+2; i++)
    in[i] = 1.;
  for (int i=0; i<nlocal; i++)
    out[i] = 0.;

  for (int step=0; step<nsteps; step++) {
#pragma omp parallel for schedule(static)
    for (int i=0; i<nlocal; i++) {
      out[i] = ( in[i]+in[i+1]+in[i+2] )/3.;
    }
#pragma omp parallel for schedule(static)
    for (int i=0; i<nlocal; i++)
      in[i+1] = out[i];
    in[0] = 0; in[nlocal+1] = 1;
  }
```

Index