Parallelism (PAR)

Data-aware task decomposition strategies
(or ... how to reduce memory coherence traffic in your parallelization)

Eduard Ayguadé, Daniel Jiménez and Gladys Utrera

Computer Architecture Department
Universitat Politècnica de Catalunya

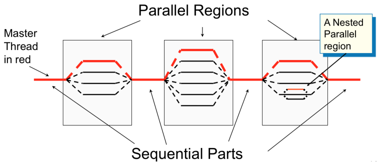
Course 2021/22 (Spring semester)

Learning material for this Unit

- Atenea: Unit 5 Data decomposition
  - Atenea quizz with motivation example
  - Going further: distributed-memory architectures video lesson (OPTIONAL)
- These slides to deep dive into the concepts in this Unit
- Collection of Exercises: problems in Chapter 5

Task creation in OpenMP (summary)

- `#pragma omp parallel`: One **implicit** task is created for each thread in the team (and immediately executed)



- `int omp_get_num_threads`: returns the number of threads in the current team. 1 if outside a parallel region
- `int omp_get_thread_num`: returns the identifier of the thread in the current team, between 0 and `omp_get_num_threads()-1`

Outline

Reducing memory coherence traffic: improving locality by data decomposition

Reducing memory coherence traffic: avoiding false sharing

Data decomposition

Data is distributed across the multiple memories in our NUMA multiprocessor system

- Remember first touch, assigning as home for a memory line the NUMA node that first touches the memory page that contains the memory line
- For simplicity we usually consider one memory line per page

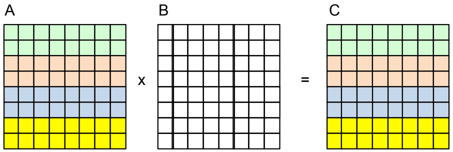
To improve locality ... the programmer should try to assign work so that tasks executed in a certain NUMA node access the data that is stored in the main memory of that NUMA node

- Use of implicit tasks created in `parallel` ...
- ... and the identifier of the thread they are running to decide what to execute

Example: matrix multiply using implicit tasks (1)

Consider that matrices A and C are distributed among 4 processors by rows (P_0 first block of rows, in green; P_1 second block of rows, in brown; P_2 third block of rows, in blue; and P_4 last block of rows, in yellow). Matrix B is not specified.

```
void matmul (double C[MATSIZE][MATSIZE],
             double A[MATSIZE][MATSIZE],
             double B[MATSIZE][MATSIZE]) {
   for (int i=0; i<MATSIZE; i++)
      for (int j=0; j<MATSIZE; j++)
         for (int k=0; k<MATSIZE; k++)
            C[i][j] += A[i][k]*B[k][j];
}
```



Assume that $thread_i$ in a parallel OpenMP regions is executed by processor P_i.

Example: matrix multiply using implicit tasks (2)

```
void matmul (double C[MATSIZE][MATSIZE],
             double A[MATSIZE][MATSIZE],
             double B[MATSIZE][MATSIZE]) {
int i, j, k;

#pragma omp parallel
   {
   int myid = omp_get_thread_num();
   int howmany = omp_get_num_threads();
   int i_start =  myid * (MATSIZE/howmany);
   int i_end = i_start + (MATSIZE/howmany);
   if (myid == howmany-1) i_end = MATSIZE;

   for (int i=i_start; i<i_end; i++)
      for (int j=0; j<MATSIZE; j++)
         for (int k=0; k<MATSIZE; k++)
            C[i][j] += A[i][k]*B[k][j];
   }
}
```

Load balancing problem: last implicit task may get up to `howmany-1` additional iterations!

Example: matrix multiply using implicit tasks (3)

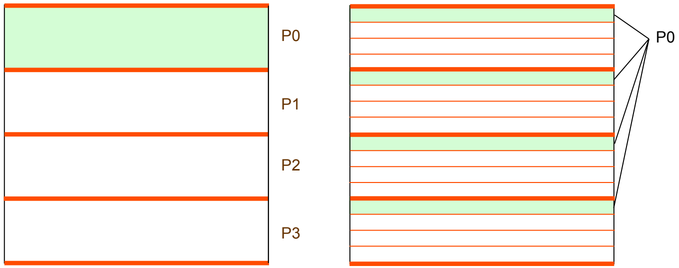
Let's reduce the load unbalance to 1 iteration at most ...

```
void matmul (double C[MATSIZE][MATSIZE],
             double A[MATSIZE][MATSIZE],
             double B[MATSIZE][MATSIZE]) {
int i, j, k;

#pragma omp parallel
   {
   int myid = omp_get_thread_num();
   int howmany = omp_get_num_threads();
   int i_start =  myid * (MATSIZE/howmany);
   int i_end = i_start + (MATSIZE/howmany);
   int rem = MATSIZE % howmany;
   if (rem != 0) {
       if (myid < rem) {
           i_start += myid;
           i_end += (myid+1);
       } else {
           i_start += rem;
           i_end += rem;
       }
   }
   ...
   }
}
```

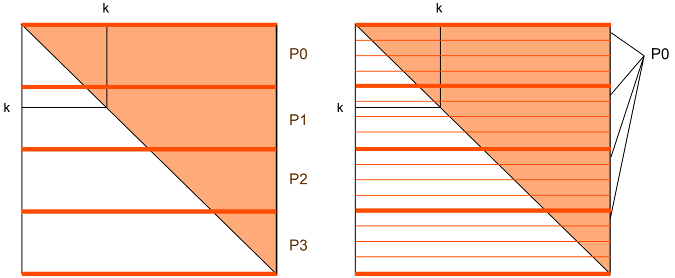
Data distributions for geometric decomposition

Block (left) and cyclic (right) data decompositions



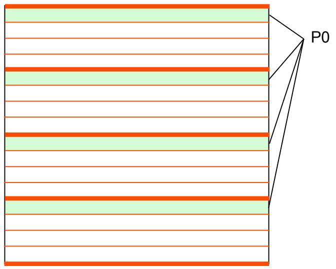
Data distributions for geometric decomposition

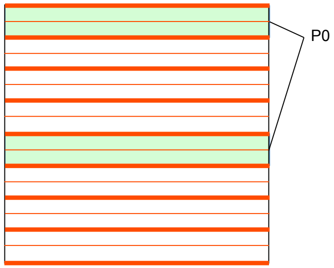
Block (left) and cyclic (right) data decompositions in a triangular iteration space



Data distributions for geometric decomposition

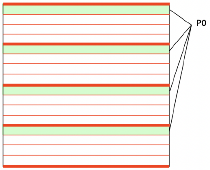
Cyclic (left) and block-cyclic (right) data decompositions





Code transformations for other data decompositions (1)

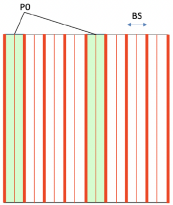
CYCLIC DATA DECOMPOSITION, by ROWS



```
#pragma omp parallel private (i, j)
{
    int myid = omp_get_thread_num();
    int howmany = omp_get_num_threads();

    for (int i=myid; i<N; i+= howmany)
        for (int j=0; j<N; j++)
    ...
}
```

BLOCK-CYCLIC DATA DECOMPOSITION, by COLUMNS

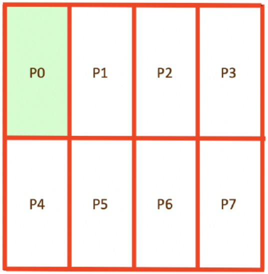


```
#define BS ...
#pragma omp parallel private (i, j)
{
    int myid = omp_get_thread_num();
    int howmany = omp_get_num_threads();

    for (int i=0; i<N; i++)
        for (int jj=myid*BS; jj<N; jj+=howmany*BS)
             for (int j=jj; j<jj+BS; j++)
        ...
}
```

Code transformations for other data decompositions (2)

2D BLOCK / BLOCK DATA DECOMPOSITION



```
#pragma omp parallel private (i, j)
{
    int my_i = omp_get_thread_num()/4;
    int my_j = omp_get_num_threads()%4;
    int BS_i = N/2;
    int BS_j = N/4;
    int i_start = my_i * BS_i;
    int i_end = i_start + BS_i;
    int j_start = my_j * BS_j;
    int j_end = j_start + BS_j;

    for (int i=i_start; i<i_end; i++)
        for (int j=j_start; j<j_end; j++)
    ...
}
```

Input, output or input–output data decomposition

- Input decomposition partitions input data structures across tasks. Output data structures are not considered for locality optimization
- Output decomposition partitions output data structures across tasks. Input data structures are not considered for locality optimization
- They can be combined if both input and output data structures benefit from the same data decomposition

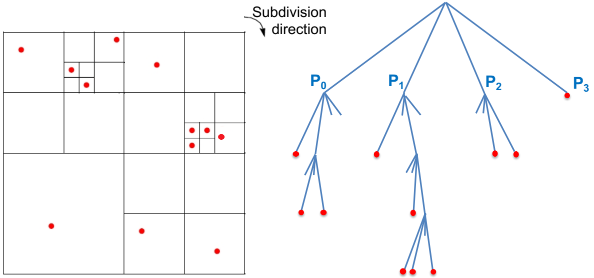
The Owner Computes rule

It defines who is responsible for doing the computations:

- In the case of output data decomposition, the owner computes rule implies that the output is computed by the task to which the output data is assigned.
- In the case of input data decomposition, the owner computes rule implies that all computations that use the input data are performed by the task to which the input is assigned.

Data distributions for recursive decomposition (Optional)

Quadtree to represent particles in an N-body problem

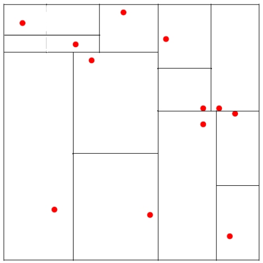


Particles in 2D space Quatree representation

- Each leaf node stores position and mass for a body
- Other nodes store center of mass and total mass for all bodies below

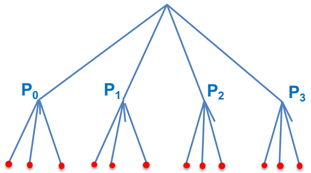
Data distributions for recursive decomposition

Orthogonal distribution of the particles of an N-body, so that in each bi-partition the number of particles in each side is halved (load balancing)



Particles in 2D space

Orthogonal partitioning:
alternating vertical and horizontal



Quatree representation

Example: N-body computation (sequential)

Sequential code

```
void main() {
 // Initialize tree
 for (t=0; t<tmax; t++) {
   for (i=0;i<N;i++) doTimeStep(tree, node[i]);  // node[i] points to body i in the tree
   // Update the positions and velocities
   // Migrate bodies if required in the tree
 }
}
```

TreeNode structure

```
typedef struct {
   ...
   char     isLeaf
   TreeNode *quadrant[2][2];
   double   F; // force on node
   double   center_of_mass[3];
   double   mass_of_center;
   ...
} TreeNode;
```

Calculate forces implementation

```
void doTimeStep(TreeNode* subTree, TreeNode* body) {
  if(subTree) {
    if(!subTree->isLeaf && !distant(subTree, body)) {
      for(int i=0; i<2; i++)
       for(int j=0; j<2; j++)
           doTimeStep(subTree->quadrant[i][j], body);
    }
    else // subtree is a leaf
      calcForces(subTree, body); // update F field for body
  }
}
```

A distant subtree is approximated as a single body with mass/center

Example: N-body computation (data decomposition)

Each thread computes the forces in each node caused by the sub-tree assigned to it

```
void main() {
   // initialize tree
   ...
   #pragma omp parallel private(subtree) num_threads(4)
   {
      // Each thread will get a subtree
      subtree = partition(tree, omp_get_thread_num(), omp_get_num_threads());
      for (int t=0; t<tmax; t++) {
          for (int i=0;i<N;i++) doTimeStep(subtree, node[i]);
          // Update the positions and velocities
          ...
          if (...) {   // Migrate bodies if required in the quad-tree
              ...
              subtree = partition(tree, omp_get_thread_num(), omp_get_num_threads());
          }
      }
   }
}
```

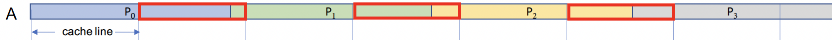
Outline

Reducing memory coherence traffic: improving locality by data decomposition

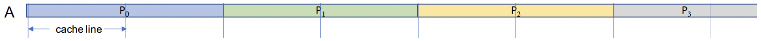
Reducing memory coherence traffic: avoiding false sharing

Examples/situations of false sharing ... (1)

```
#pragma omp parallel
{
    int myid = omp_get_thread_num();
    int BS = n / omp_get_num_threads();
    for (i=myid*BS; i<(myid+1)*BS; i++) A[i] = foo(i*23);
}
```

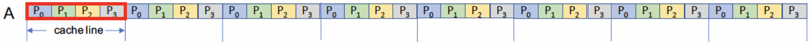


Possible solution: introduce some load unbalance, so that BS corresponds with a number of elements that fit in a number of complete cache lines

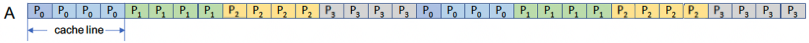


Examples/situations of false sharing ... (2)

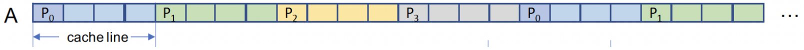
```
#pragma omp parallel
{
    int myid = omp_get_thread_num();
    int howmany = omp_get_num_threads();
    for (i=myid; i<n; i+=howmany) A[i] = foo(i*23);
}
```



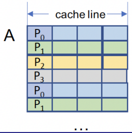
Possible solution: make larger chunk size (p.e. 4) → block-cyclic



Alternative solution: Add padding – i.e. one element per cache line



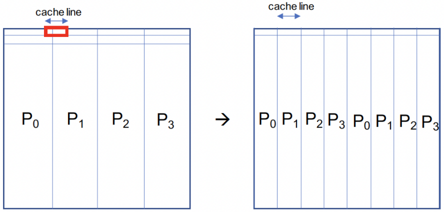
How? `int A[100];` → `A[100][4];`
And the access needs to change ...
`A[i][0] = foo(i*3);`



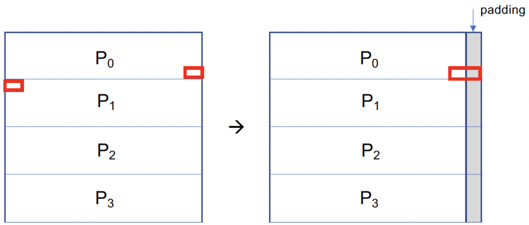
Examples/situations of false sharing ... (3)

In 2D matrices we can also have false sharing problems ... solutions ?

- block → block-cyclic



- Add some padding



Parallelism (PAR)

Data-aware task decomposition strategies
(or ... how to reduce memory coherence traffic in your parallelization)

Eduard Ayguadé, Daniel Jiménez and Gladys Utrera

Computer Architecture Department
Universitat Politècnica de Catalunya

Course 2021/22 (Spring semester)