

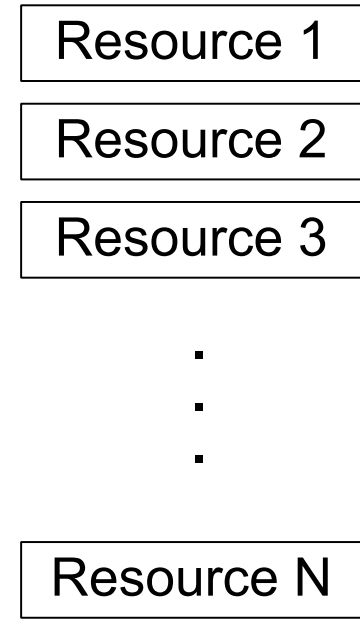
# StarSs roadmap

**BSC**

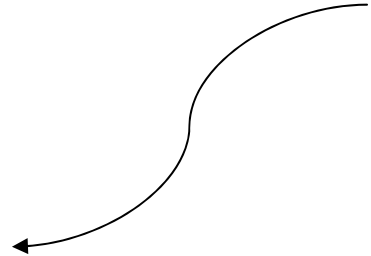
# ing mod

Parallel Resources  
(multicore, SMP, cluster, gr

## • Basic idea



Synchronization,  
results transfer

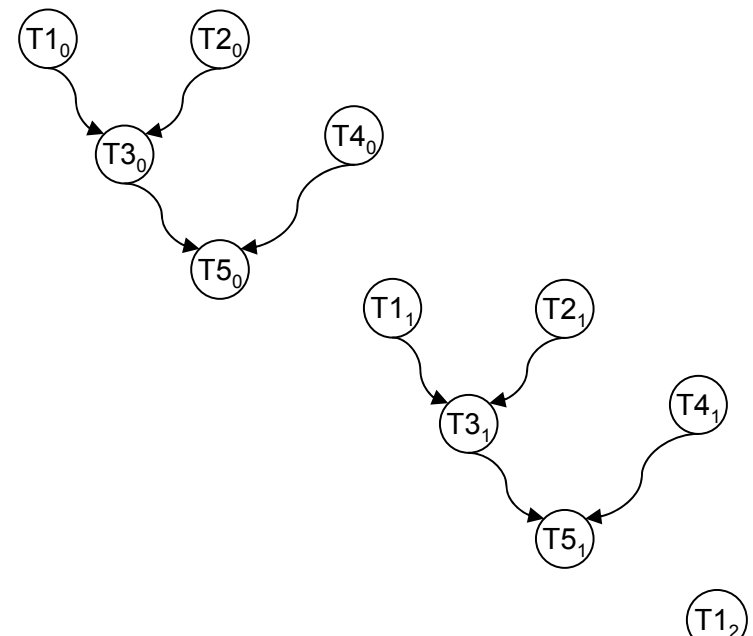
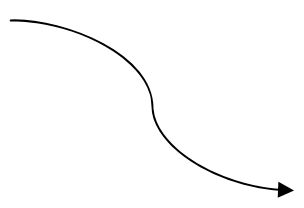


Task selection +  
parameters direction  
(input, output, inout)

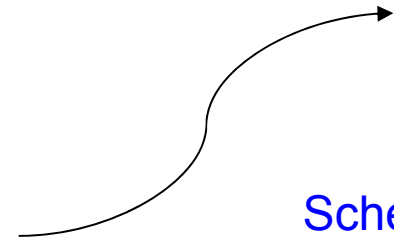
## Sequential Application

```
...  
for (i=0; i<N; i++){  
    T1 (data1, data2);  
    T2 (data4, data5);  
    T3 (data2, data5, data6);  
    T4 (data7, data8);  
    T5 (data6, data8, data9);  
}  
...
```

Task graph creation  
based on data  
precedence



Scheduling,  
data transfer,  
task execution



# ing mod

- GRIDSs,  
COMPSs

- Tailored for  
Grids or  
clusters

- Data  
dependency  
analysis based  
on files

- C/C++, Java

- SMPSs

- Fortran

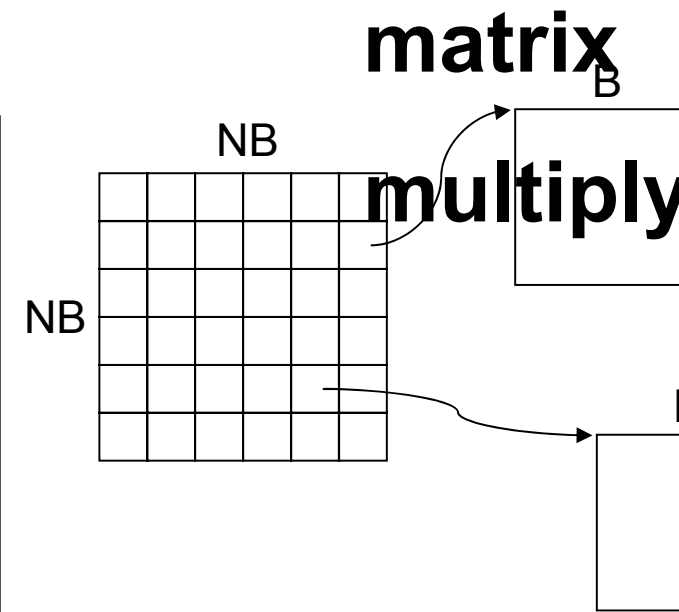
```

main (int argc, char **argv) {
    int i, j, k;

    initialize(A, B, C);

    for (i=0; i < NB; i++)
        for (j=0; j < NB; j++)
            for (k=0; k < NB; k++)
                block_addmultiply( C[i][j], A[i][k], B[k][j]);
}

```



Block algorithms

No side effects

```

static void block_addmultiply( float C[BS][BS], float A[BS][BS],
float B[BS][BS]) {
    int i, j, k;

    for (i=0; i < B; i++)
        for (j=0; j < B; j++)
            for (k=0; k < B; k++)
                C[i][j] += A[i][k] * B[k][j];
}

```

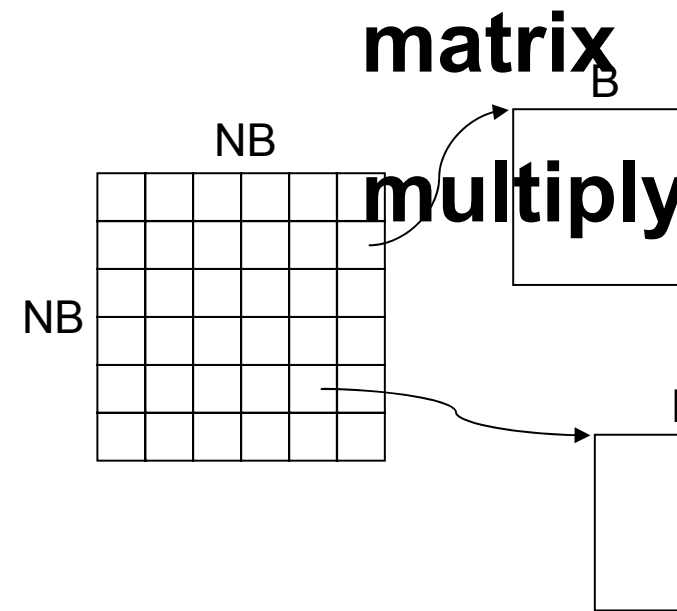
```

main (int argc, char **argv) {
    int i, j, k;

    initialize(A, B, C);

    for (i=0; i < NB; i++)
        for (j=0; j < NB; j++)
            for (k=0; k < NB; k++)
                block_addmultiply( C[i][j], A[i][k], B[k][j]);
}

```



Block algorithms

No side effects

```

#pragma cxx task input(A, B) inout(C)
static void block_addmultiply( float C[BS][BS], float A[BS][BS],
float B[BS][BS]) {
    int i, j, k;

    for (i=0; i < B; i++)
        for (j=0; j < B; j++)
            for (k=0; k < B; k++)
                C[i][j] += A[i][k] * B[k][j];
}

```

## work

- Reducing overheads in the runtime – `addTask`
- Scheduling for locality for CellSs but also for SMPSs
- Accessing array

# OpenMP

## 3.0 and

- SMPSs +  
**SMPSs**  
OpenMP:

the best of  
both world

- SMPSs:  
data-  
dependence

```
for ( i = 0; i < N; i++ ) {  
#pragma omp task input(a) output(a)  
{  
a = f(i,a);  
}  
}
```

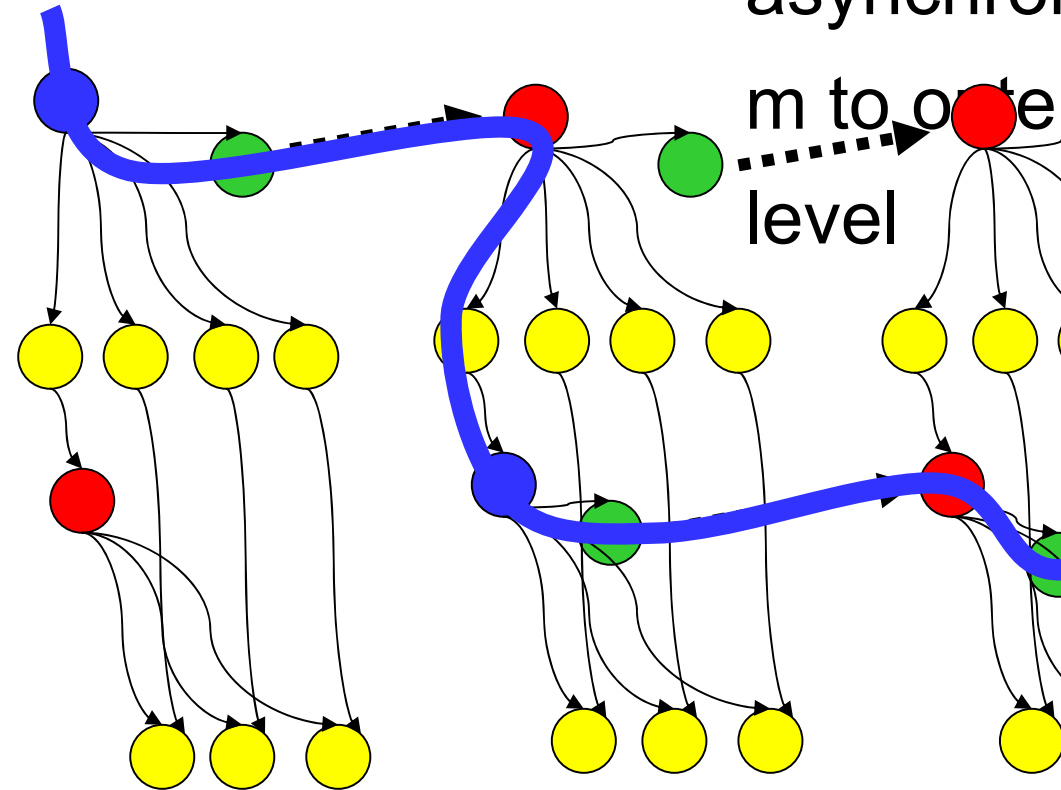
analysis, tas  
dependency  
graph  
scheduling,  
renaming

```

k=0; k<N; k++) {
  (mine) {
Factor_panel(A[k]);
send (A[k])
else {
receive (A[k]);
if (necessary) resend (A[k]);

r (j=k+1; j<N; j++)
update (A[k], A[j]);

```



- Extend asynchronous to ordered level

```

css task inout(A[SIZE])
ctor_panel(float *A);
css task inout(A[SIZE]) inout(B[SIZE])
date(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])

```

- Overlap communication and computation

```

#pragma css task input(A[SIZE]) output(send_req)]
void Irecv(float *A, int *send_req)]
{
    MPI_Isend (A,...);
}

```

```

#pragma css task input(send_req) inout(A{SIZE})]
void Wait_Irecv(int *send_req, float *A)]
{
    int ierr, go;
    ierr = MPI_Test(send_req,&go,...)]
    if(go==0) #pragma css restart
    ierr = MPI_Wait(send_req,...);
}

```

- Asynchronous MPI calls + wait tasks

```

#pragma css task output(recv_req,A[SIZE])]
void Irecv(int *recv_req, float *A)]
{
    MPI_Irecv(A,...)]
}

```

```

#pragma css task input(recv_req) inout(A[SIZE])]
void Wait_Irecv(int *recv_req, float *A)]
{
    int ierr, go;
    ierr = MPI_Test(recv_req,&go,...)]
    if(go==0) #pragma css restart
    ierr = MPI_Wait(recv_req,...);
}

```

- Restart the task
- Avoid

ing

Models

will become

more and

more

complex,

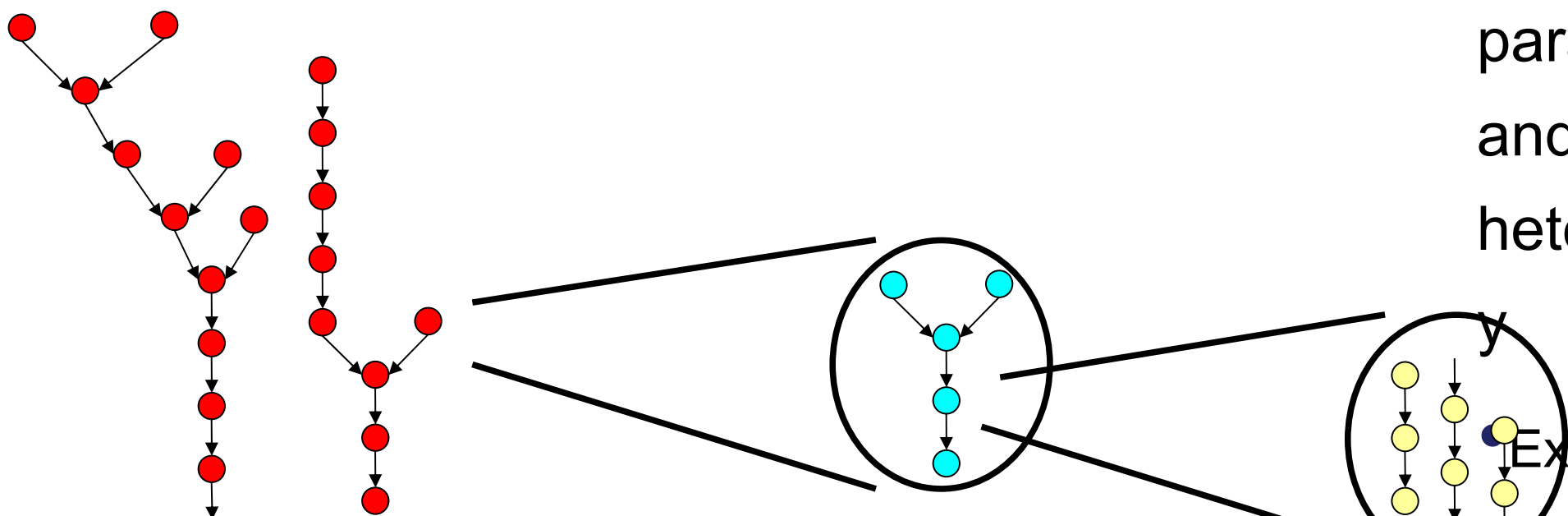
with different

levels of

parallelism

and

heterogeneous



Example: