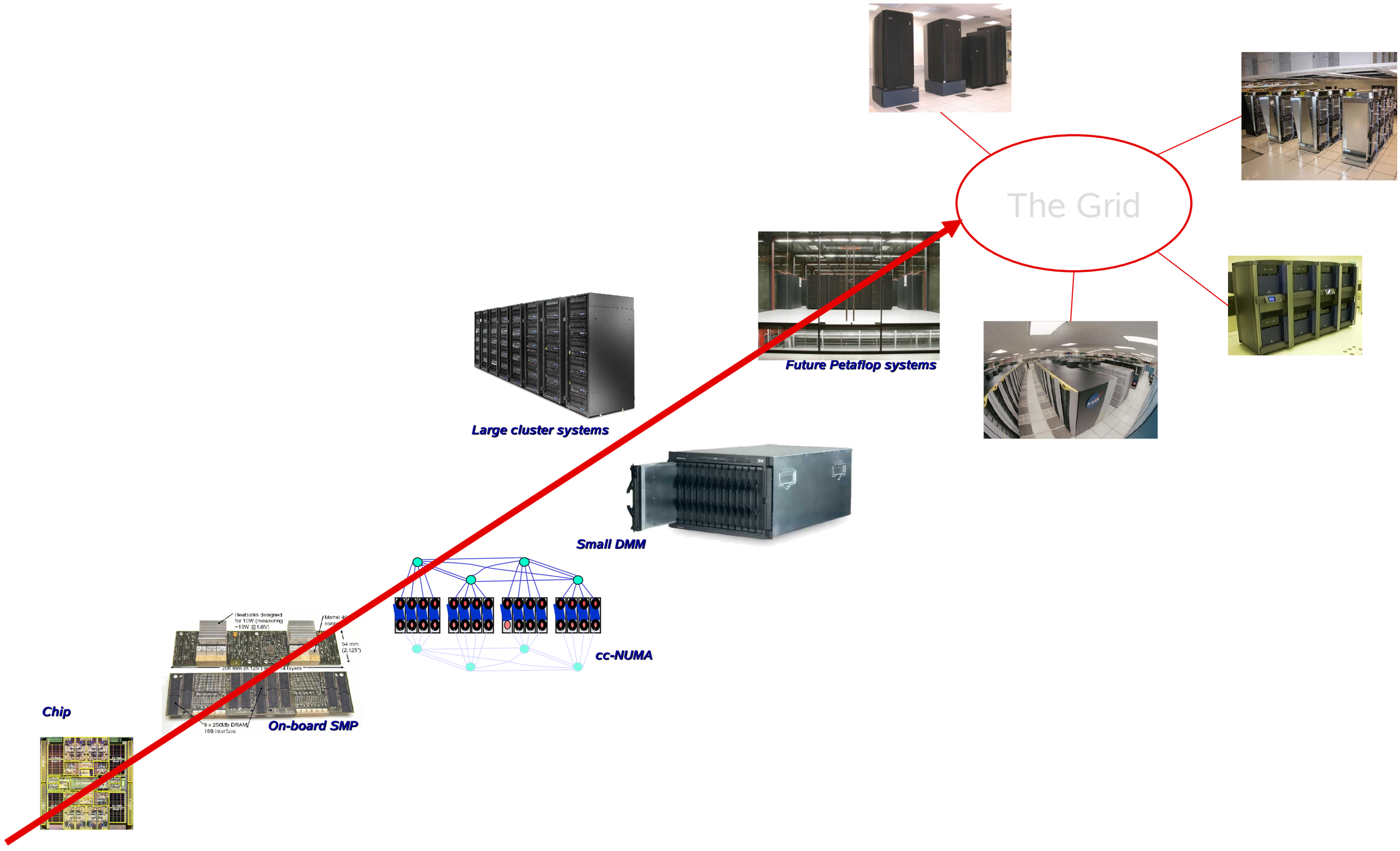
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Programming the landscape of parallel resources

Rosa M. Badia

BSC

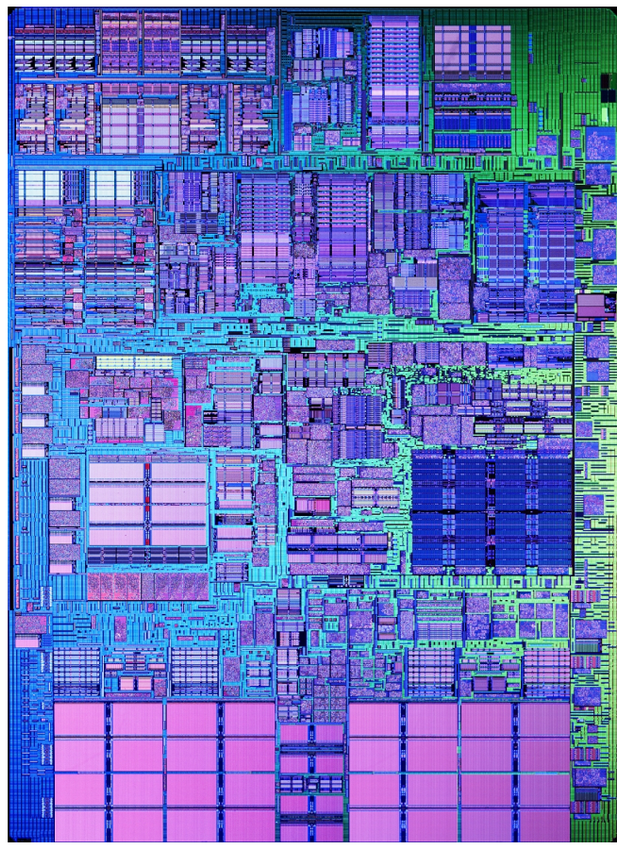
Motivation


The Grid
Future Petaflop systems
Large cluster systems
Small DMM
cc-NUMA
Chip
On-board SMP
Heatsinks designed for 15W (measuring ~13W @1.6V)
54 mm (2.125")
9 x 256Mb DRAM; 16B interface

Outline

- StarSs programming model
  - GRIDSs/COMPSs
  - CellSs/SMPSs
- Composition of programming models
  - SMPSs + MPI
  - SMPss + CellSs
  - GRIDSs + MPI + OpenMP/SMPSs @ MareNostrum
  - GRIDSs + MPI + CellSs @ MariCel
- STARSs Users' Applications
- Issues/Ongoing work
- Conclusions

STARSs programming model



- Superscalar processor
  - Instructions
  - Functional units
  - Registers
  - Memory
- Flow sequential program
- Concurrent execution, out of order, speculation, ...

STARSs programming model

- Basic idea

Parallel Resources
(multicore,SMP, cluster, grid)

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 selection +

parameters direction

(input, output, inout)

Synchronization,

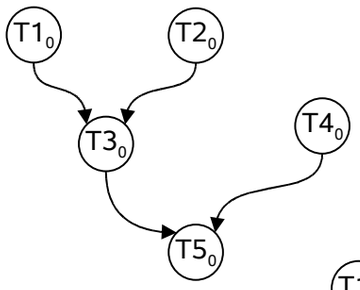
results transfer

Resource 1

Resource 2

Resource 3

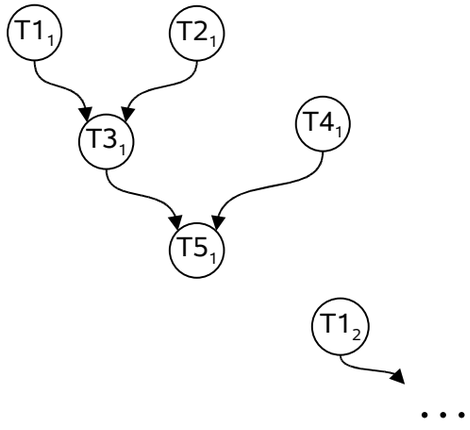
Resource N



Task graph creation

based on data

precedence



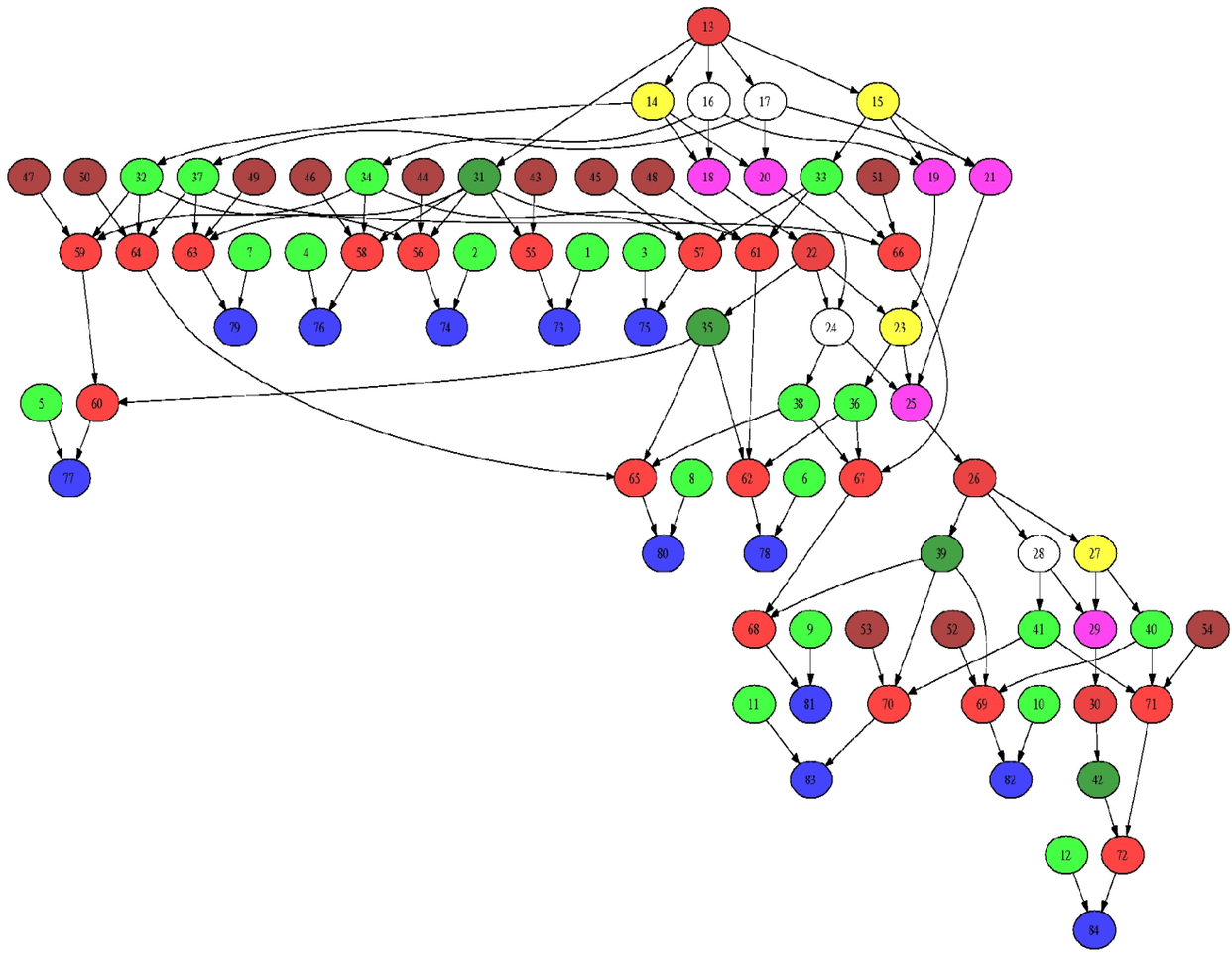
Scheduling,

data transfer,

task execution

STARSs programming model

- Main objective: Reduce the complexity of applications development
  - Complexity of writing an application for a parallel platform comparable to writing it for a sequential platform
- Main characteristics
  - Task: unit of parallel work
  - Non intrusive programming model
  - Data dependence detection
  - Data renaming
  - Exploitation of distant parallelism

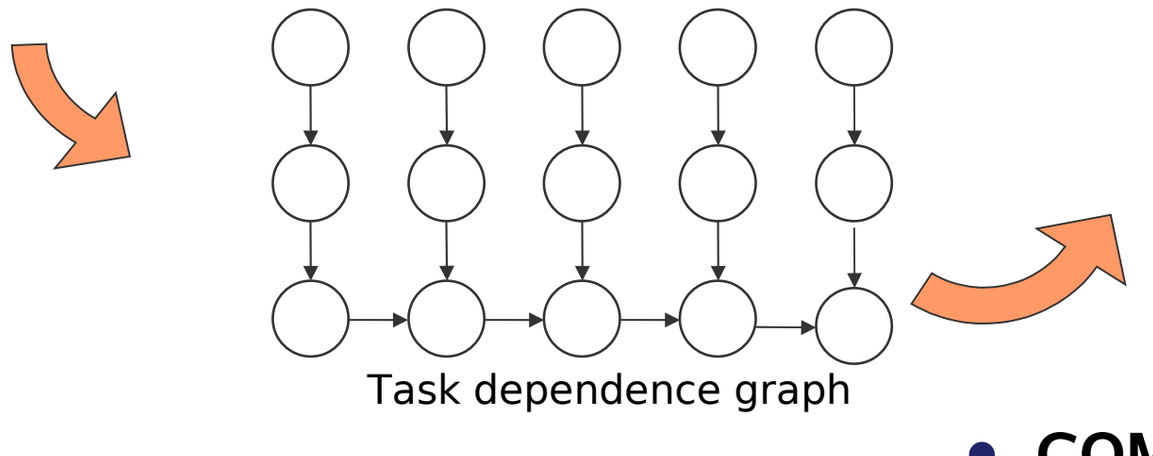


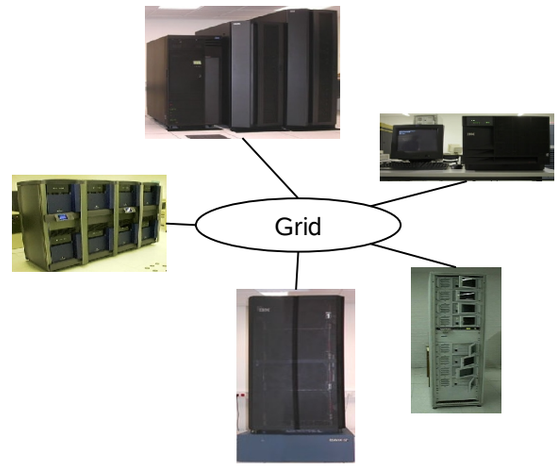
StarSs programming model

- GRIDSs, COMPSs
  - Tailored for Grids or clusters
  - Data dependence analysis based on files
  - C/C++, Java
- SMPSs
  - Tailored for SMPs or homogeneous multicores
  - Altix, JS21 nodes, Power5, Intel-Core2
  - C or Fortran
- CellSs
  - Tailored for Cell/B.E. processor
  - C or Fortran

GRIDSs/COMPSs

```
for (int i = 1; i <= 5; i++) {
  subst (refCfg, i*10000, newCfg);
  dimemas (newCfg, trace, dimOut);
  post (i*10000, dimOut, totals);
}
```





- **GRIDSs**:
  - Bindings to C/C++ and Java
  - Language:
    - IDL specify tasks + direction of arguments
    - small API
  - Based on glue code-generation
  - Data dependence unit: file
  - Transparent file transfers
  - Fault tolerance and checkpointing
  - Middleware: Globus, DRMAA, GAT, Grid-RPC, ssh/scp

- **COMPSs**:
  - Java based
  - Language:
    - No modifications
    - Optional small API
    - Tasks annotated in a Java interface
  - Based on code interception (Javassist)
  - Data dependence unit: file
  - Based on component infrastructure (GCM) – components distribution
  - Middleware: Java-GAT

GRIDSs sample program

IDL file

```
interface OPT {
    void Filter (in File referenceCFG, in double latency, in double bandwidth, out File newCFG);
    void Dimemas (in File cfgFile, in File traceFile, out File DimemasOUT);
    void Extract (in File cfgFile, in File DimemasOUT, inout File resultFile);
};
```

Parameter metadata

Main program

```
GS_On();
for (int i = 0; i < MAXITER; i++) {
    newBWd = GenerateRandom();
    Filter (referenceCFG, newBWd, newCFG);
    Dimemas (newCFG, traceFile, DimemasOUT);
    Extract (newCFG, DimemasOUT, FinalOUT);
}
fd = GS_FOpen(FinalOUT, R);
printf("Results file:\n"); present (fd);
GS_FClose(fd);
GS_Off(0);
```

COMPSs sample program

Java application

```
initialize(f1);
for (int i = 0; i < 2; i++) {
      genRandom(f2);
      add(f1, f2);
}
print(f2);
```

Java interface

```
public interface SumItf {
    @ClassName("example.Sum")
    @MethodConstraints(OSType = "Linux")
    void genRandom(
      @ParamMetadata(type = Type.FILE, direction = Direction.OUT)
      String f
    );

    @ClassName("example.Sum")
    ...
}
```

Implementation

Task constraints

Parameter metadata

CellSs/SMPSs

- Pragma based programming model: programmer specifies tasks (functions) and direction of arguments
  - `#pragma css task input ( …) output (…) inout (….)`
    `{function-definition|funtion-declaration}`
- Source to source compiler + runtime libraries
- Dependences based on task parameters
  - Scalars
  - Blocks of data
  - Structures
- Portability: sequential, SMP, homogeneous multicore, Cell, …
- Constraints
  - Blocked algorithms, task granularity
  - Tasks can only access function arguments and local data

CellSs/SMPSs syntax

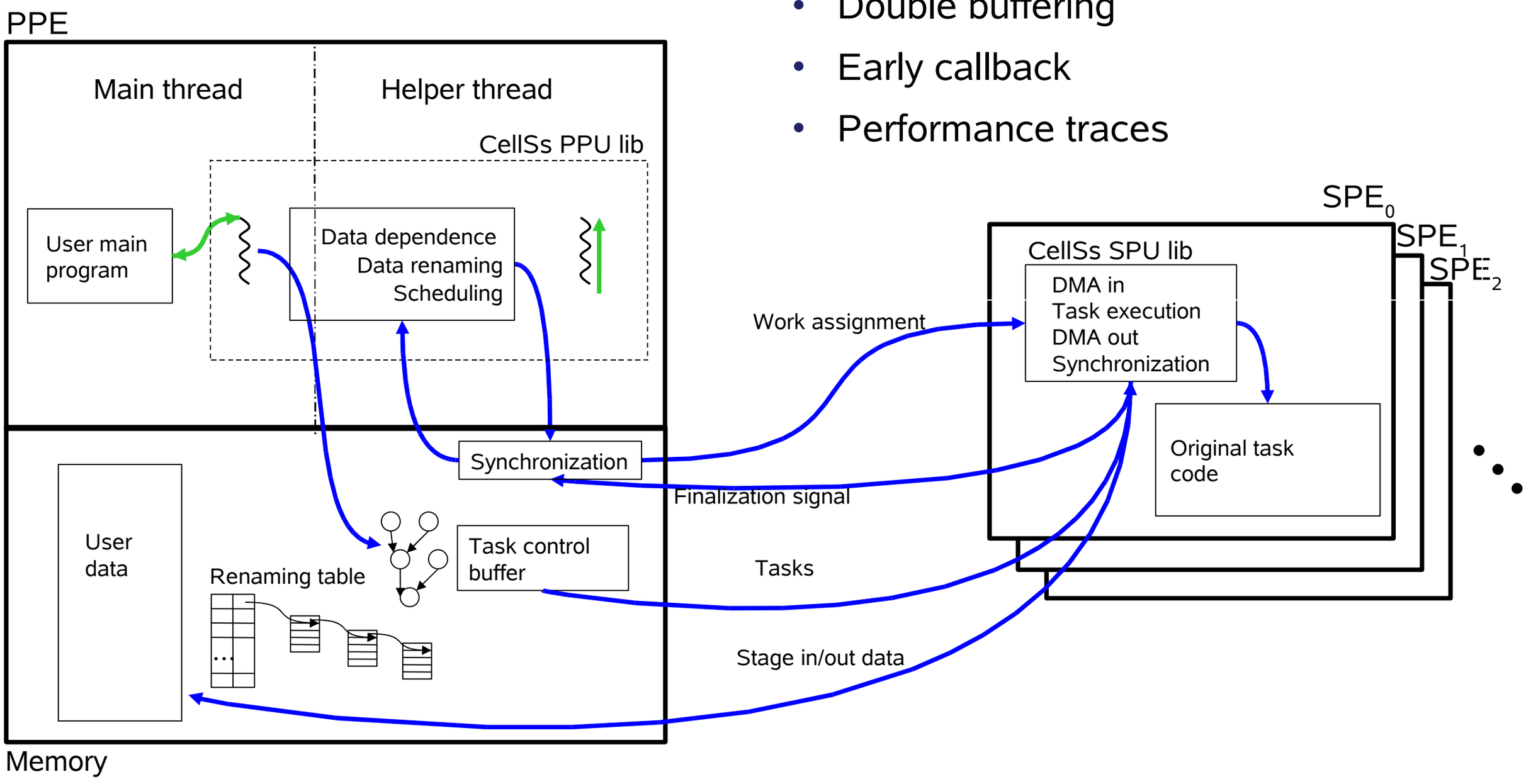
```
#pragma css task input(A, B) inout(C)
static void block_addmultiply( float C[BS][BS], float A[BS][BS], float B[BS][BS]);
```

```
#pragma css task inout(diag[B][B]) highpriority
void lu0(float *diag);
#pragma css task input(diag[B][B]) inout(row[B][B])
void bdiv(float *diag, float *row);
```

```
interface
    !$CSS TASK
    subroutine velocity(BSIZE, ii, jj, xi, yi, zi, xj, yj, zj, vx, vy, vz)
       implicit none
       integer, intent(in) :: BSIZE, ii, jj
       real, intent(in), dimension(BSIZE) :: xi, yi, zi, xj, yj, zj
       real, intent(inout), dimension(BSIZE) :: vx, vy, vz
    end subroutine
   !$CSS TASK
    subroutine v_mod(BSIZE, v, vx, vy, vz)
       implicit none
       integer, intent(in) :: BSIZE
       real, intent(out) :: v(BSIZE)
       real, intent(in), dimension(BSIZE) :: vx, vy, vz
    end subroutine
end interface
```

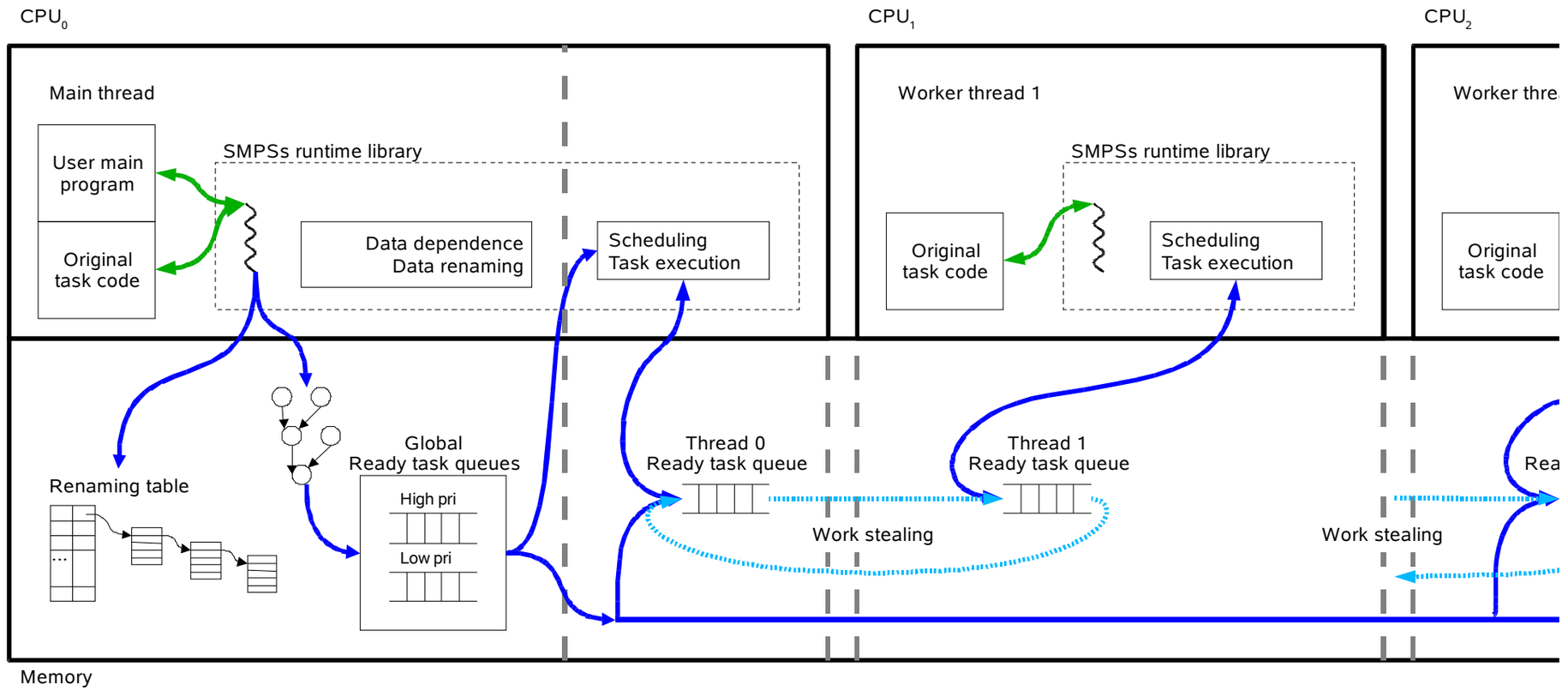
CellSs: Runtime

- Scheduling of chains/clusters of tasks
- Double buffering
- Early callback
- Performance traces



SMPSs: runtime

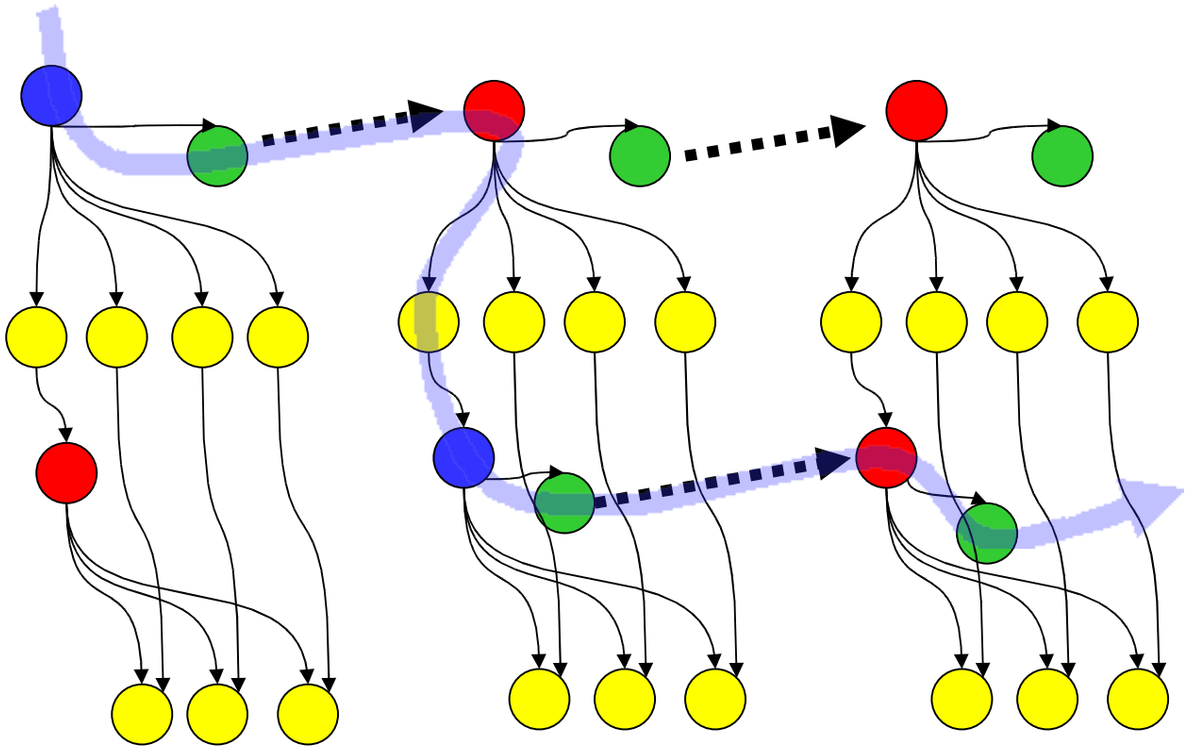
- Threads perform self-scheduling
- Each thread has private ready list
- Performance traces



SMPSs hybrid with MPI

- Extend asynchronism to outer level

```
…
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);
```

SMPSs hybrid with MPI

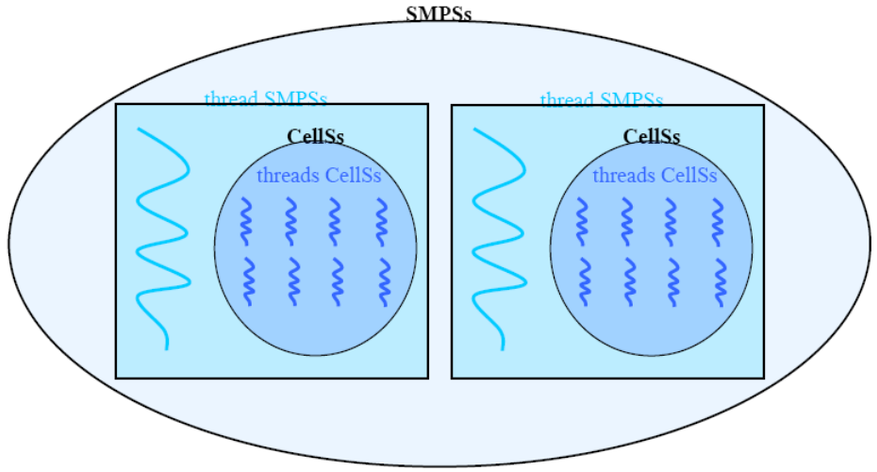
- Overlap communication and computation
  - Asynchronous/immediate MPI_calls + wait tasks
- Restartable task
  - Avoid deadlock
  - Avoid inefficient use of resources (busy wait)

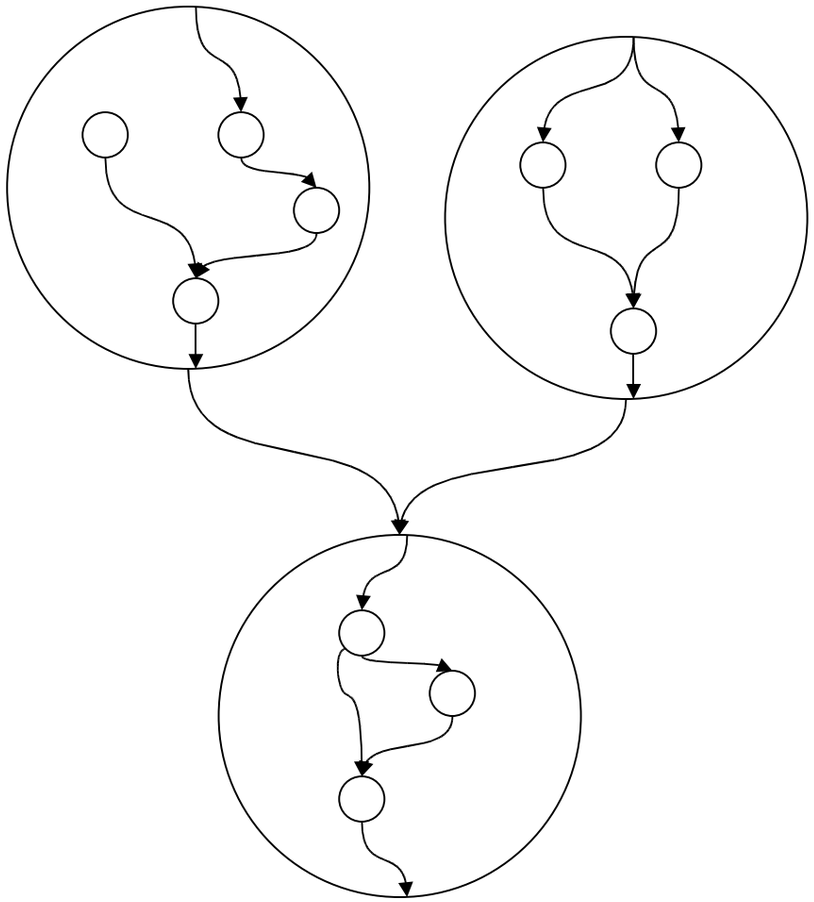
```
#pragma css task input(A[SIZE]) output(send_req)
void Isend(float *A, int *send_req)
{
   MPI_isend (A,…);
}

#pragma css task input(send_req) inout(A{SIZE])
void Wait_Isend(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,…);
}
```

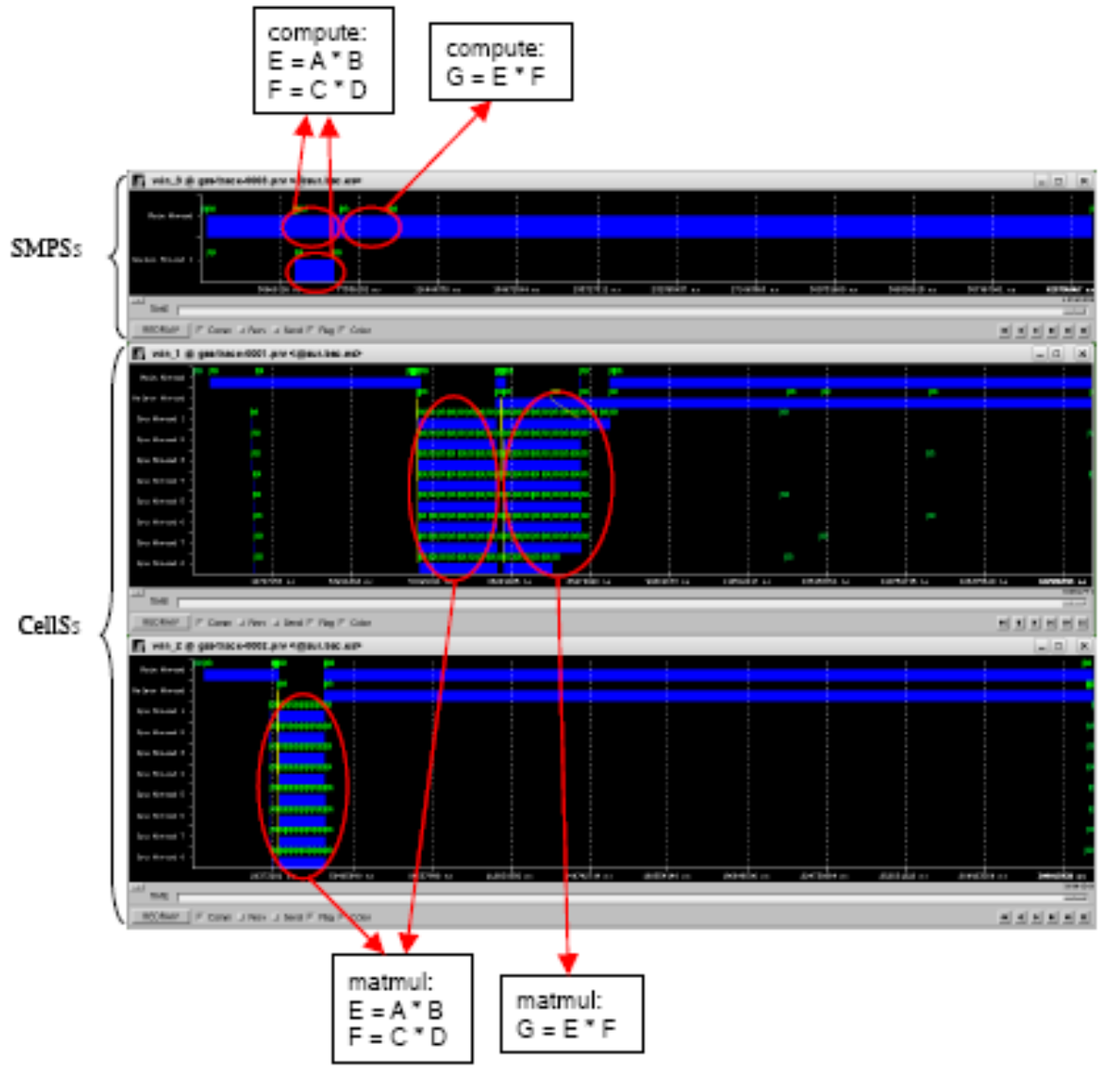
```
#pragma css task output(recv_req,A[SIZE])
void Ireceive(int *recv_req, float *A)
{
   MPI_Irecv(A,…)
}

#pragma css task input(recv_req) inout(A[SIZE])
void Wait_Ireceive(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,…);
}
```

SMPSs hierarchical

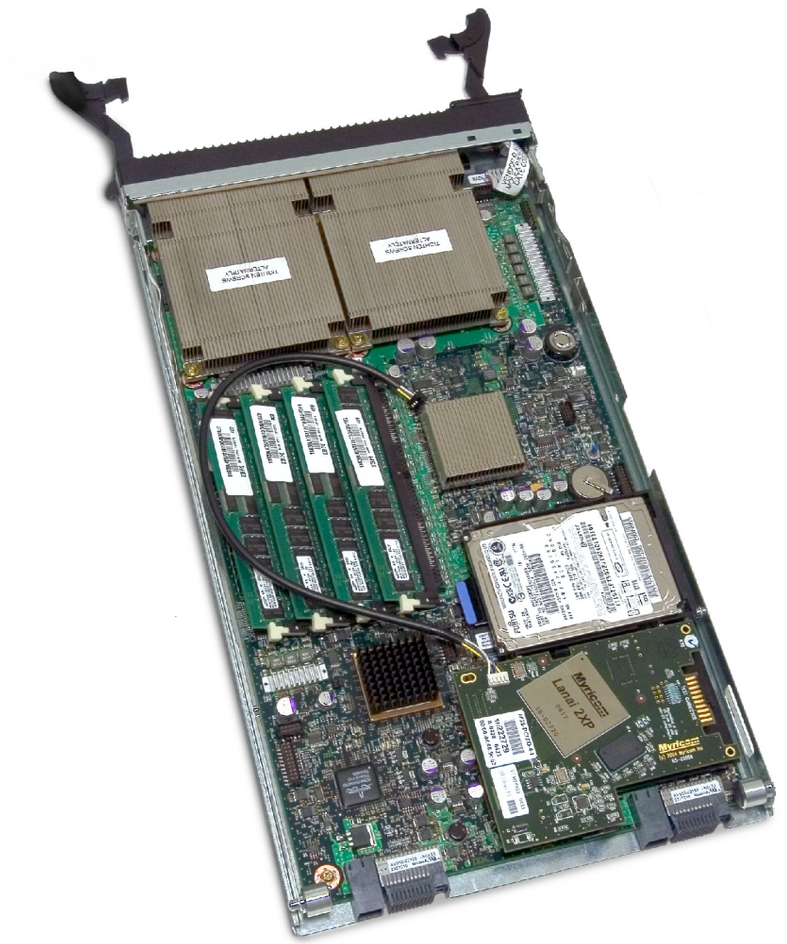
- Ideally: allow nesting at different levels of an application
- Current ongoing effort: SMPSs + CellSs





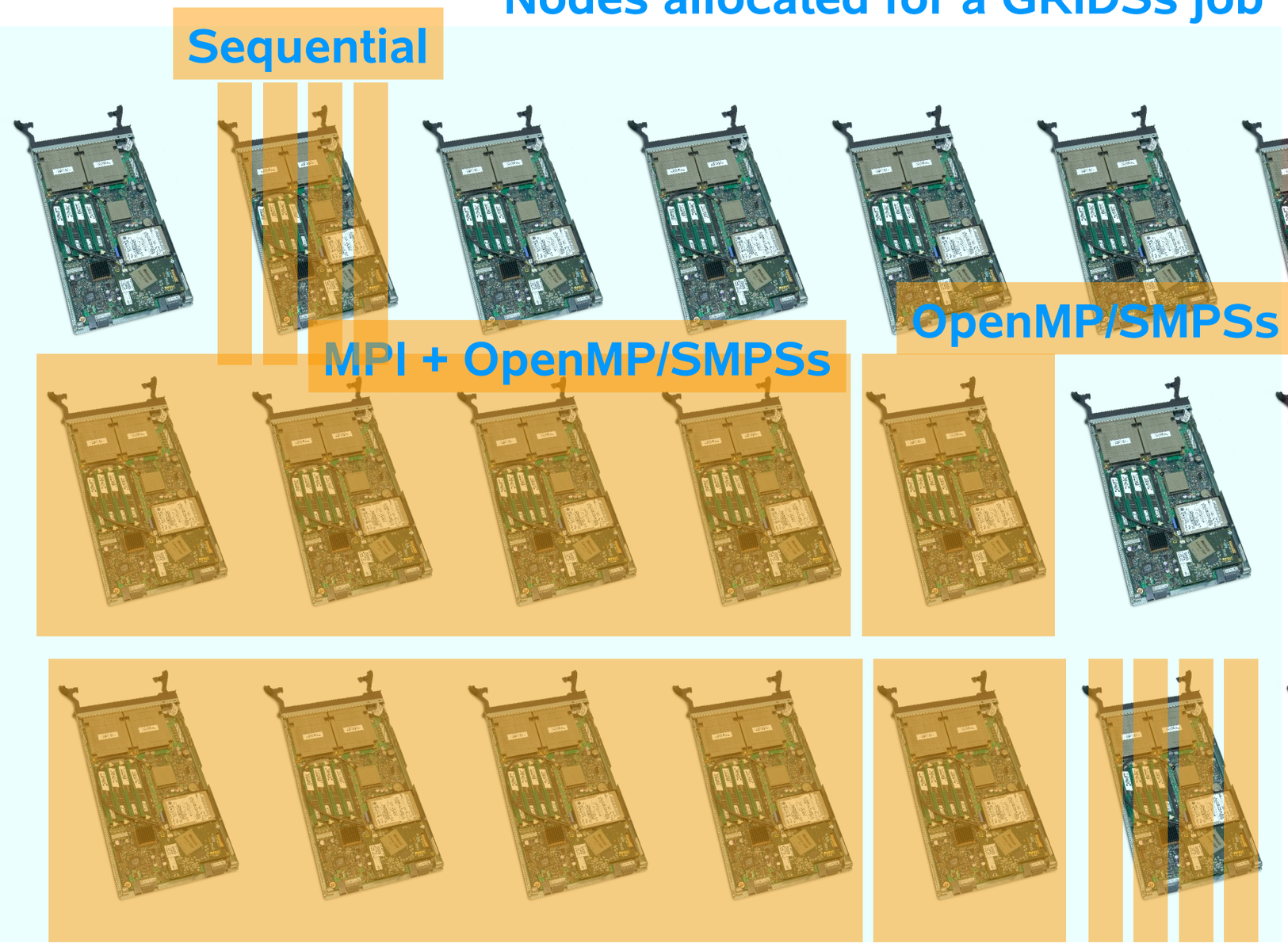


MareNostrum architecture



- 2560 JS21 2.3 GHz nodes
  - 2 dual-core PPC970MP chips
- 20 TB of Memory
- 8 GB per node
- 380 TB Storage Capacity
- 3 networks
  - Myrinet
  - Gigabit
  - 10/100 Ethernet

GRIDSs + MPI + OpenMP/SMPSs @ MareNostrum

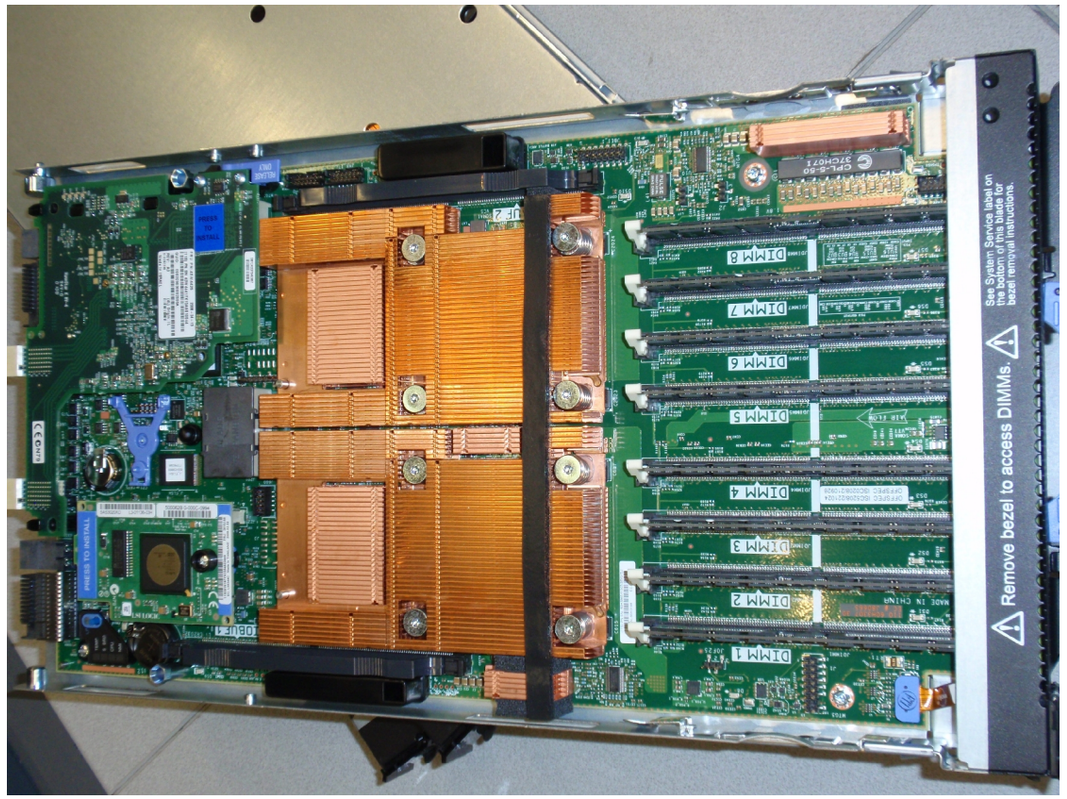


GRID superscalar tasks

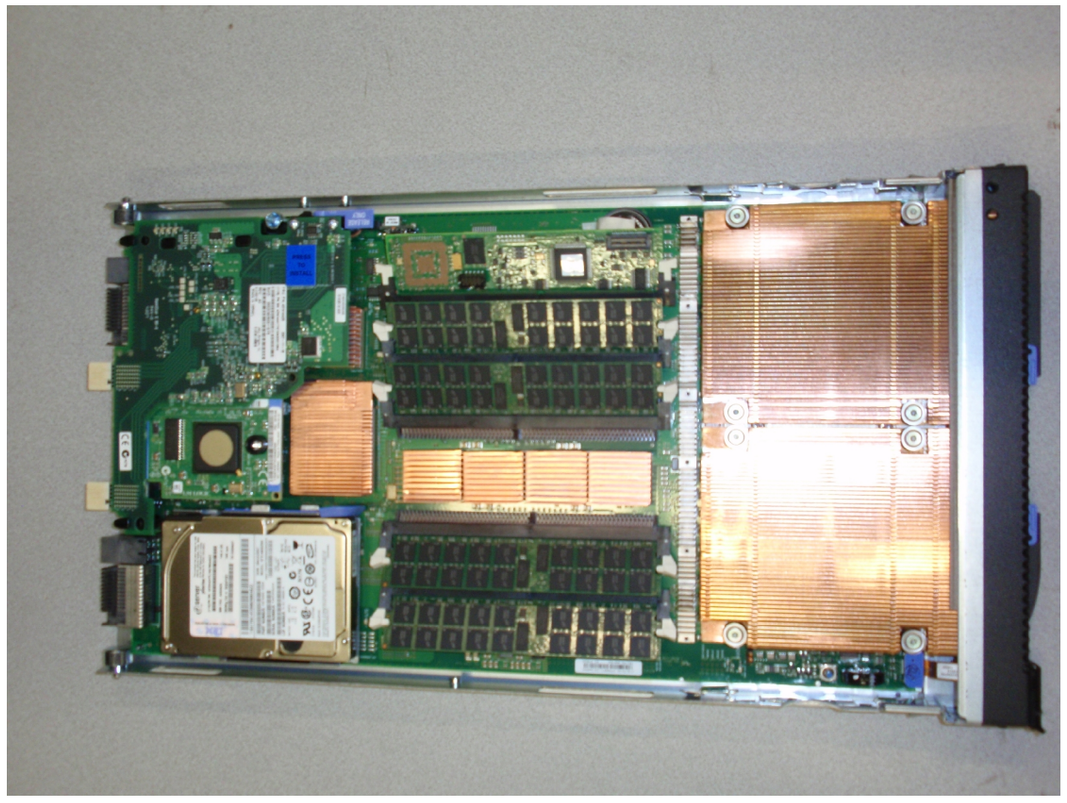
- Supported hot removal of nodes due to failure/maintenance
- Exploits GFPS, scratch file system in the nodes
- Based on ssh/scp

MariCel Architecture

QS22



JS22



- Hybrid prototype
  - 12 JS22 IBM Blades (Power6)
  - 72 QS22 (PowerXCell)
- Hypernode organization
  - 1JS22 + 6 QS22
- Infiniband 16Gb
- 14.4TF

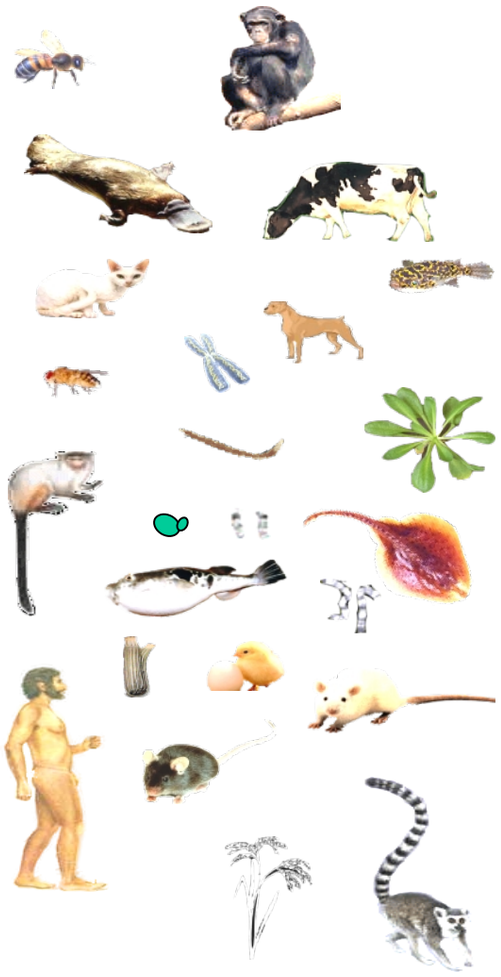
GRIDSs + MPI + OpenMP/CellSs @ MareNostrum

GRIDSs runtime

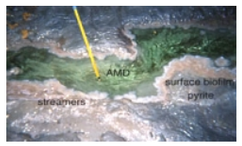
MPI + OpenMP/CellSs

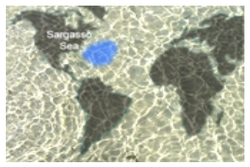
GRIDSs applications: Analysis of protein and function diversity on earth

Organisms

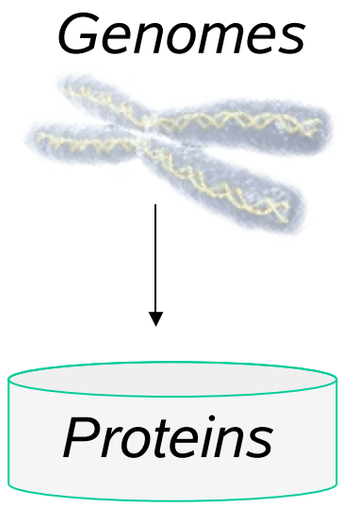


Environmental genomics

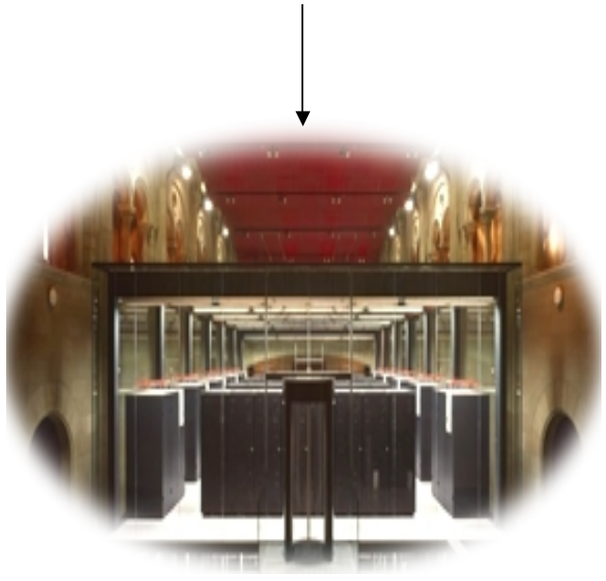


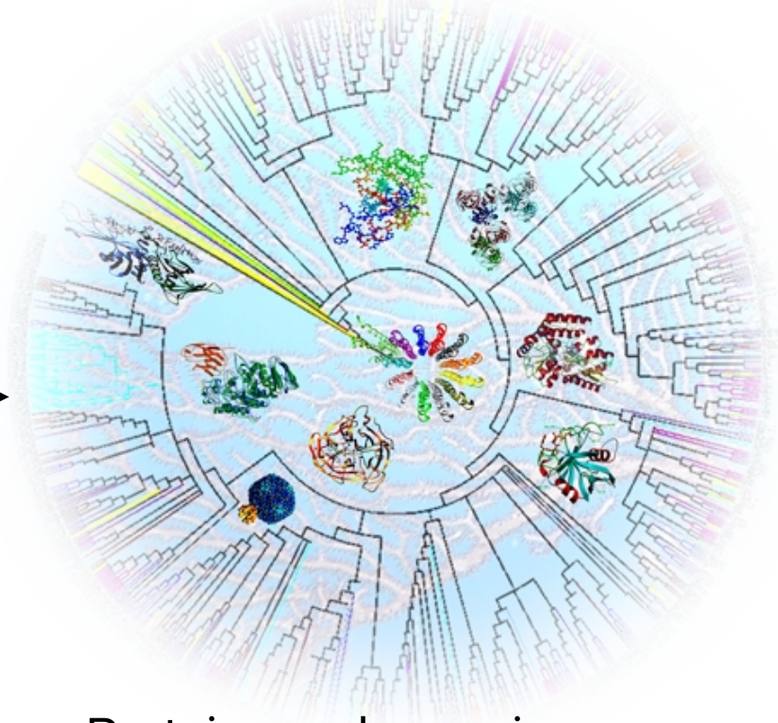


Largest protein comparison and classification done so far



- 15 million protein sequences
- BLAST orchestrated by GRIDSs
- Query and Database file size: ~ 5 Gb
- 4000 CPUs (= 1000 exclusive nodes)
- Total CPU time: 311,112 hours
- 5 Tb of results
- 100,000 tasks in each GRIDSs run



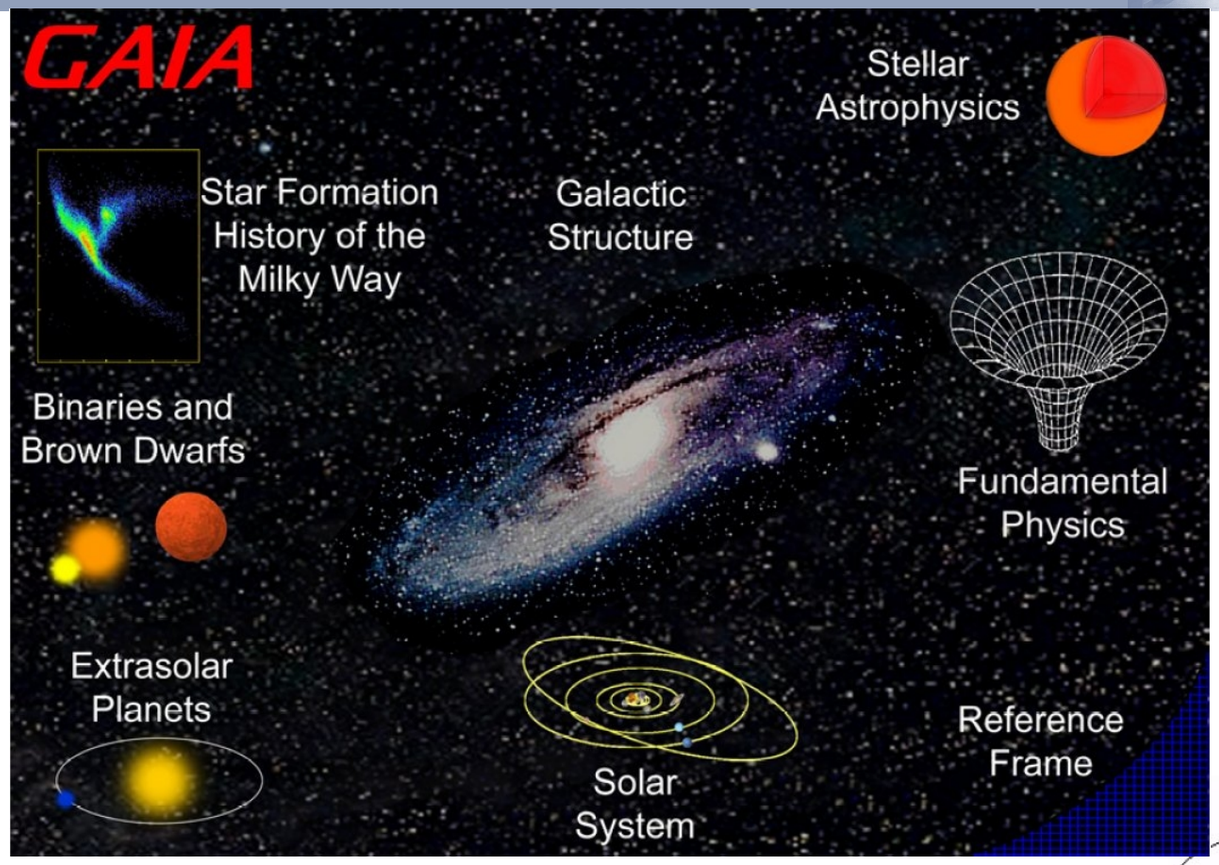


Proteins and organisms classification

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

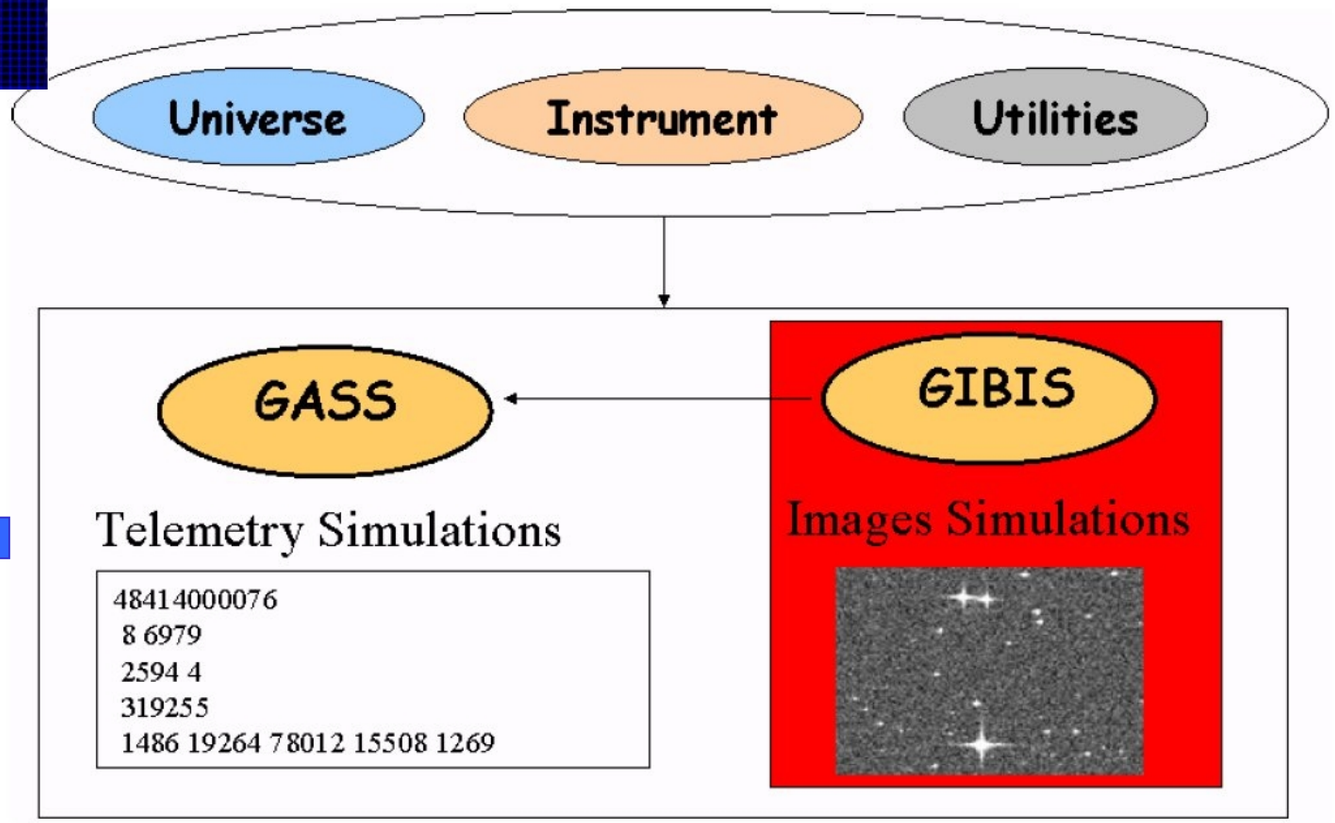
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

GRIDSs applications: Looking at the Milky Way origin



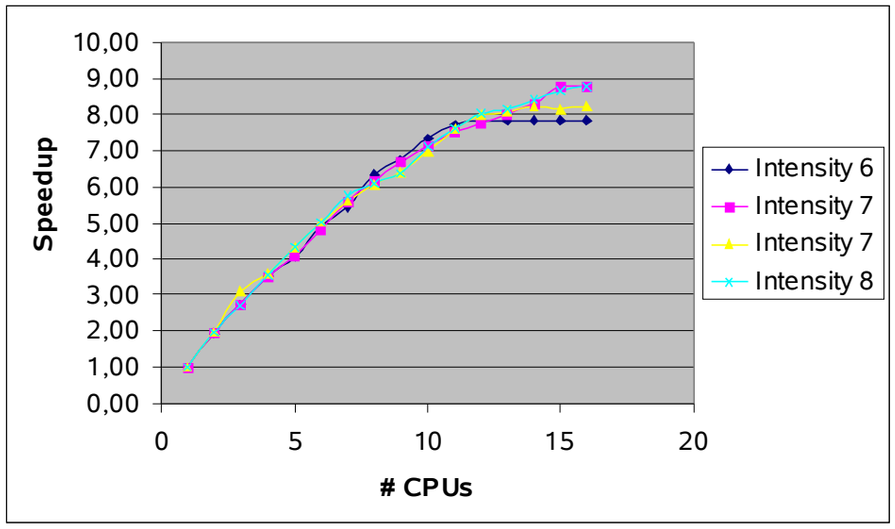
- Cycle 4
  - 20 datasets (GOG & GASS)
  - About 3.5TB of (compressed) data generated, around 20,000 zip files
  - About 350,000 CPU hours used in the generation
- Organized with several GRIDSs runs of 1000-4000 tasks each

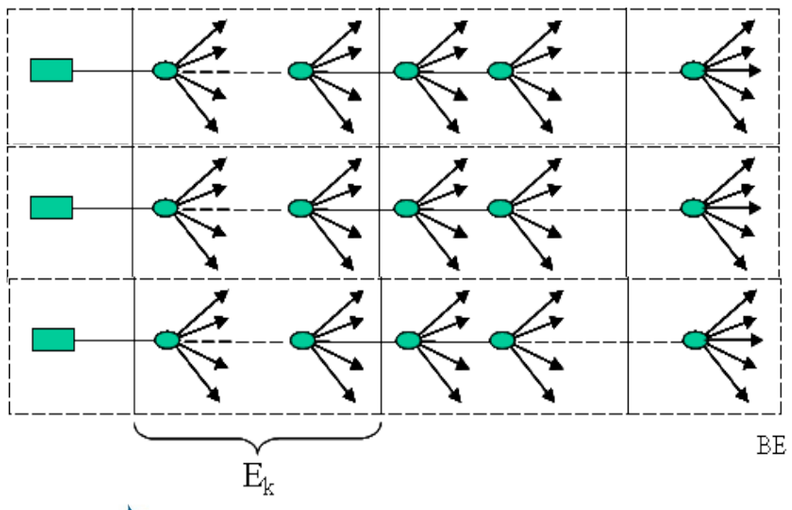
UNIVERSITAT DE BARCELONA

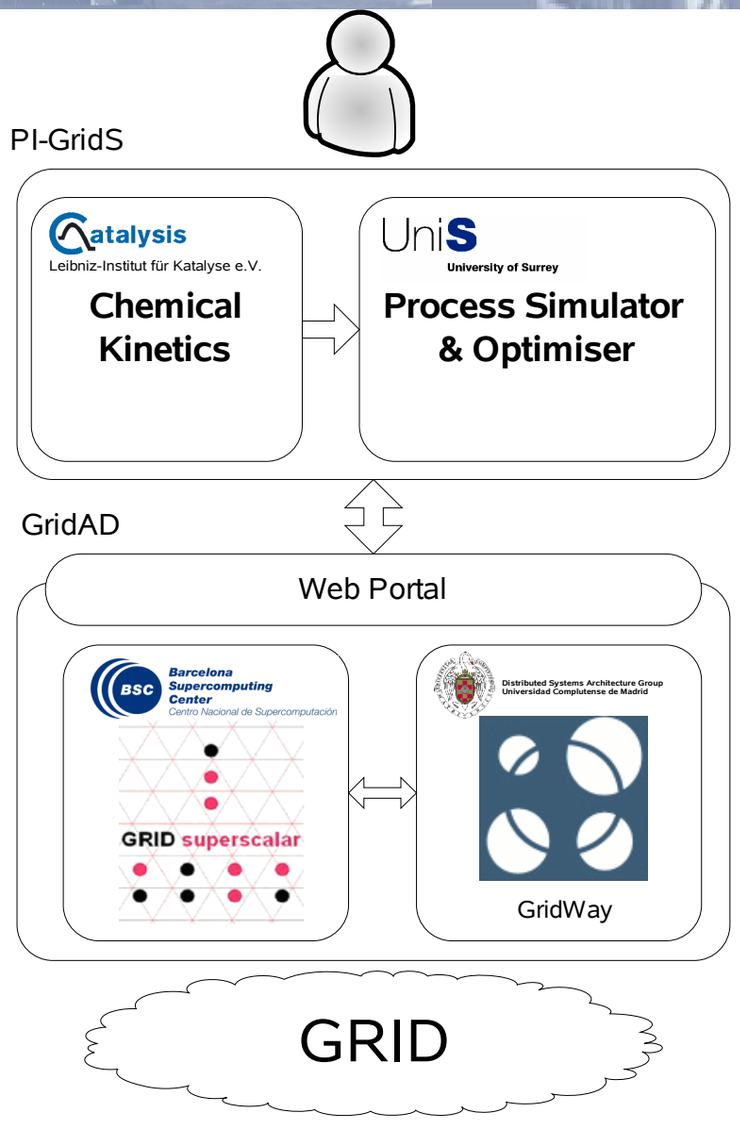


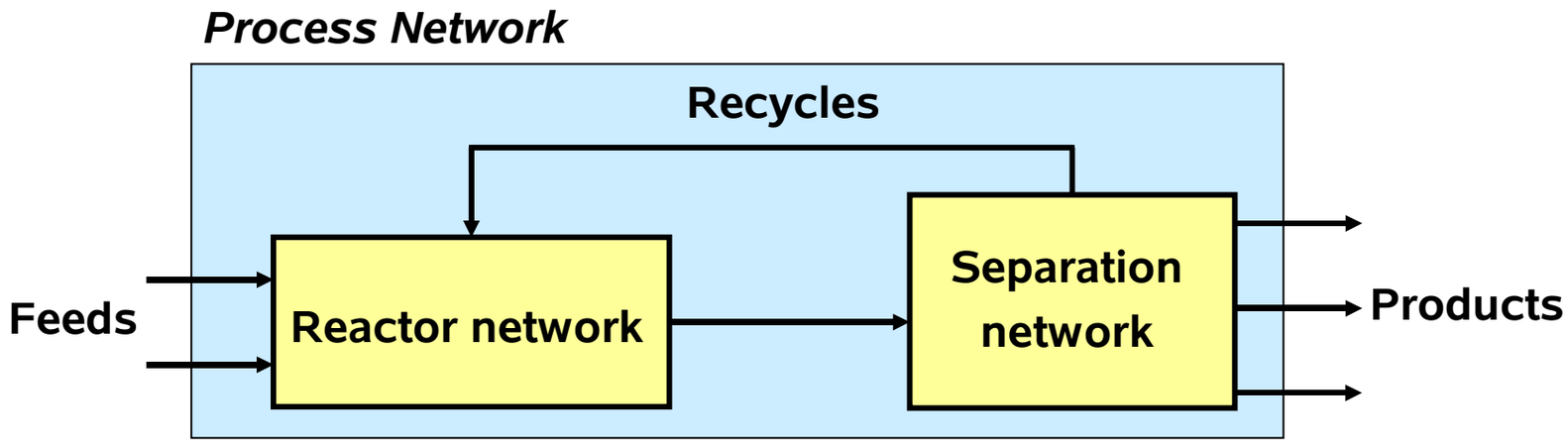
GRIDSs applications: BEinGRID BE14 - Design of products and processes

- Applied to the of acetic acid production process
- Around 150,000 tasks per execution
- Multisite runs: Barcelona, Madrid & Surrey

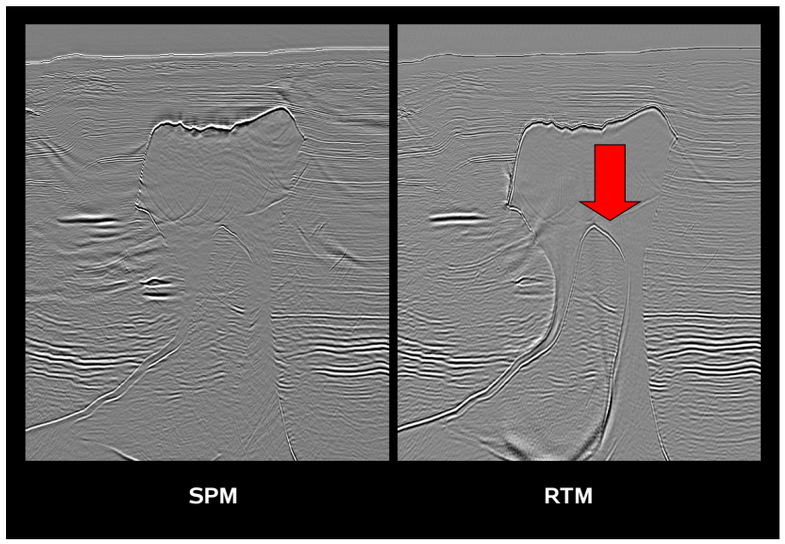


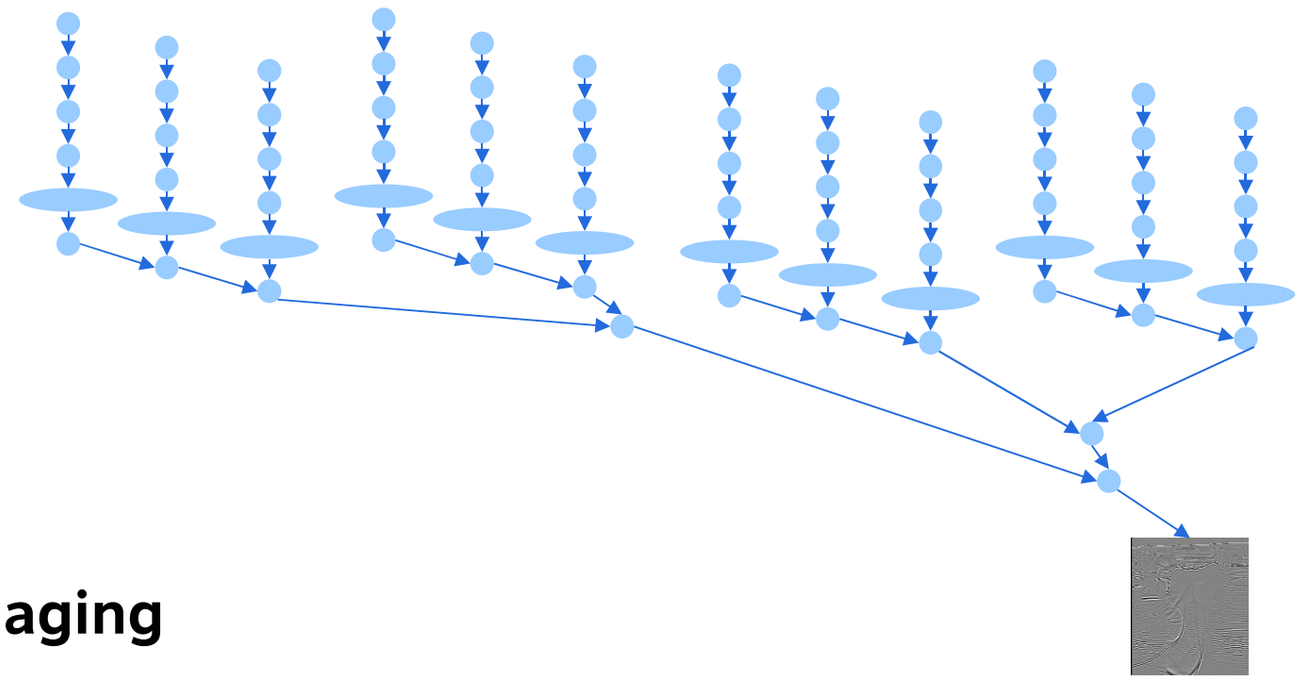






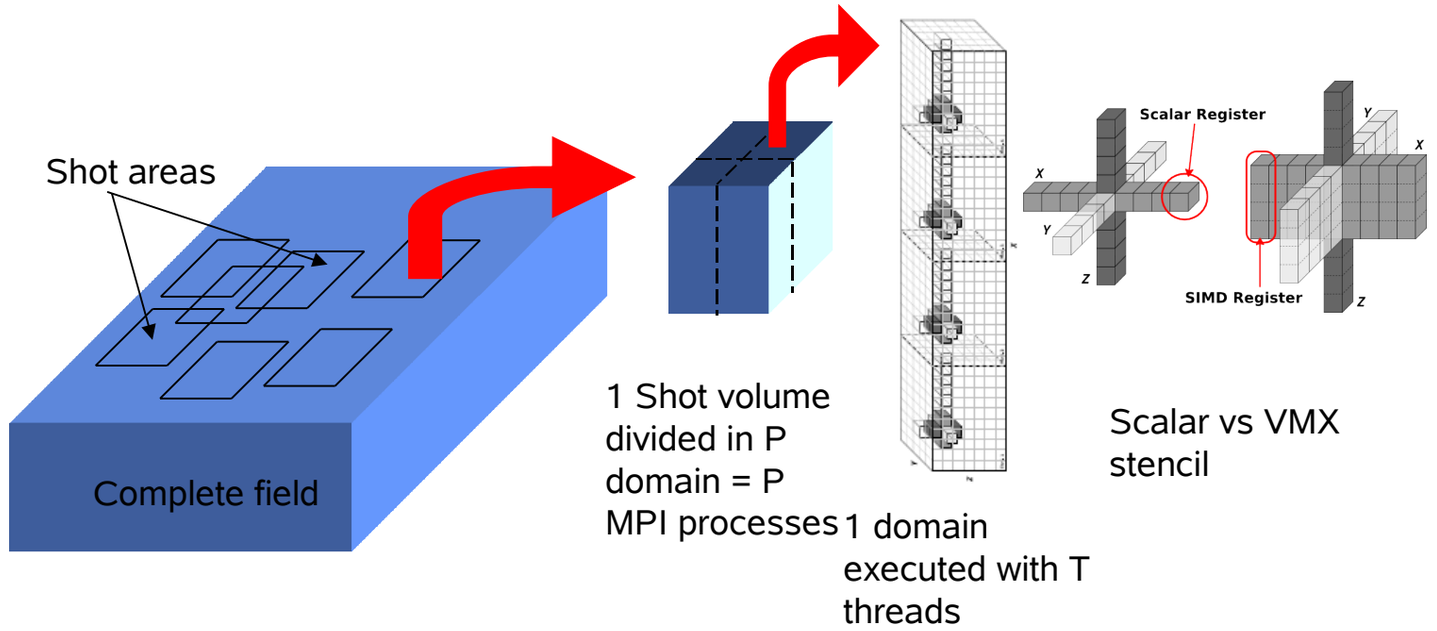
GRIDSs + MPI + OpenMP/CellSs: Reverse Time Migration





Only RTM produces proper subsalt imaging
Computationally more intensive

- 1 GRID superscalar application per image
  - 350,000 – 500,000 tasks per image
- Domain Decomposition (MPI) to process one shot between several blades
- @ MareNostrum: OpenMP inside nodes
- @ MariCel: CellSs inside nodes



REPSOL

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Issues/Ongoing work

- GRIDSs/COMPSs
  - Integration with Unicore 6 in DEISA
  - Support to the Spanish Supercomputing Network (RES) infrastructure
    - Automatic selection of node
    - Migration of GRIDSs from one to another node transparent to the user
  - Further support for combination with SMPSs/CellSs
    - Reductions, commutativity
  - Further research on benefits of components: reconfiguration, distribution, …
  - Solving issues with JVM: persistent workers?
  - Fulfillment of SLAs
  - Towards SOA
  - Implementation of HMMPfam on top of COMPSs

Issues/Ongoing work

- CellSs/SMPSs
  - Access to array regions/subobjects
  - Access to global memory by tasks
    - Blocks larger than Local Store
  - Reductions
  - Convergence with OpenMP 3.0
    - Separation dependences/data transfer
    - Nesting/Hierarchical task graph
- Scheduling for locality
- Bypasses between SPUs
- Overlays
- Control of renaming: limit size of renaming memory, lazy, on LS
- Debugging methodology and tools

Conclusions

- Task based programming model looks a good approach
- However
  - Is not the only solution
    - Should co-exist with others: MPI, streaming, TM, …
    - Must converge with OpenMP
  - *Automatism* is not always well accepted
    - Need to educate programmers
- Overheads in architectures like Cell make it difficult … although more fun
  - Current issues can be solved in forthcoming devices' generations
- Need for ways of expressing the hierarchy of resources in the programming model
  - Must for many-cores and heterogeneous devices

STARSs websites

- Contact
  - rosa.m.badia@bsc.es
- GRIDSs
  - www.bsc.es/grid/gridsuperscalar
- CellSs
  - www.bsc.es/cellsuperscalar
- SMPSs
  - www.bsc.es/smpsuperscalar
- All of them available for download (open source, Apache v2, GPL and LGPL)

- Thanks to:
  - Eduard Ayguadé
  - Pieter Bellens
  - Jose M. Cela
  - Jorge Ejarque
  - Isaac Jurado
  - Jesus Labarta
  - Xavier Luri
  - Luis Martinell
  - Josep M. Perez
  - Judit Planas
  - Sebastian Reyes
  - David Torrents
  - Raul Sirvent
  - Enric Tejedor