Supporting parallel tasks with GRID superscalar

Jorge Ejarque, Enric Tejedor, Daniele Lezzi,
Raül Sirvent, Rosa M. Badia
Barcelona Supercomputing Center (BSC-CNS)
Universitat Politècnica de Catalunya (UPC)
Consejo Superior de Investigaciones Cientificas (CSIC)

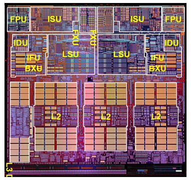
IBERGRID Conference 2010

Outline

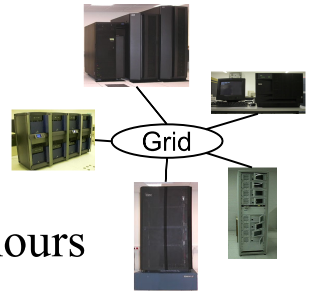
- GRID superscalar overview
- Extensions for supporting parallel tasks
- Usage Examples
- Conclusions

GRID superscalar overview

- Reduce the development complexity of Grid applications to the minimum
  - Writing an application for a computational Grid may be as easy as writing a sequential application
- Basic idea:



$\cong$



ns ➡ seconds/minutes/hours

- Target applications: composed of tasks, most of them repetitive
  - Granularity of the tasks of the level of simulations or programs

GRID superscalar overview

- GRID superscalar components:
  - User interface (programming environment)
    - Interface Definition Language (IDL) file
    - Main program
    - Subroutines/functions
    - Constraints file
  - Runtime
  - Automatic code generator (generate stubs, scripts, …)
- Supported Programming languages:
  - C/C++, Perl, Java

GRID superscalar overview

- Interface Definition Language (IDL) file
  - In/Out/InOut files or scalars
  - The functions listed will be executed in a remote node in the Grid.

```
interface OPT {
void subst ( in File referenceCFG, in double latency, in double bandwidth, \
    out File newCFG );
void dimemas ( in File cfgFile, in File traceFile, in double goal, \
    out File DimemasOUT );
void post ( in double bw, in File DimemasOUT, inout File resultFile );
void display ( in File resultFile );
};
```

- Master code

```
GS_On();
for (int i = 0; i < MAXITER; i++) {
   newBWd = GenerateRandom();
   subst (referenceCFG, newBWd, newCFG);
   dimemas (newCFG, traceFile, DimemasOUT);
   post (newBWd, DimemasOUT, FinalOUT);
   if(i % 3 == 0) display(FinalOUT);
}
fd = GS_FOpen(FinalOUT, R);
printf("Results file:\n"); present (fd);
GS_FClose(fd);
GS_Off(0);
```

GRID superscalar overview

- Subroutines/functions

```
void dimemas(char *newCFG, char *traceFile, char *DimemasOUT)
{
   char command[500];
   putenv("DIMEMAS_HOME=/usr/local/cepba-tools");
   sprintf(command, "/usr/local/cepba-tools/bin/Dimemas -o %s %s",
         DimemasOUT, newCFG );
   GS_System(command);
}
```

```
void display(char *toplot)
{
   char command[500];
   sprintf(command, " ./display.sh %s", toplot);
   GS_System(command);
}
```

```
void subst(char *f1, char *f2, char *fout){
FILE *fp;
int i,j,k;
for (i=1; i<1000; i++)
        for (j=0; j<1000; j++)
                k= j%i;
fp = fopen(fout,"w");
fprintf(fp,"Call to concat(%s, %s, %s)\n", f1, f2, fout);
fclose(fp);

}
```

Constraints file

- Constraints and cost functions

```
void dimemas_constraints(char *newCFG, char *traceFile)
{
        return "(member(\"Dimemas23\", other.SoftNameList))“;
}

double dimemas_cost(char *newCFG, char *traceFile) {
        double time;
        time = (GS_Filesize(traceFile)/1000000) * GS_Gflops();
        return(time);
}
```

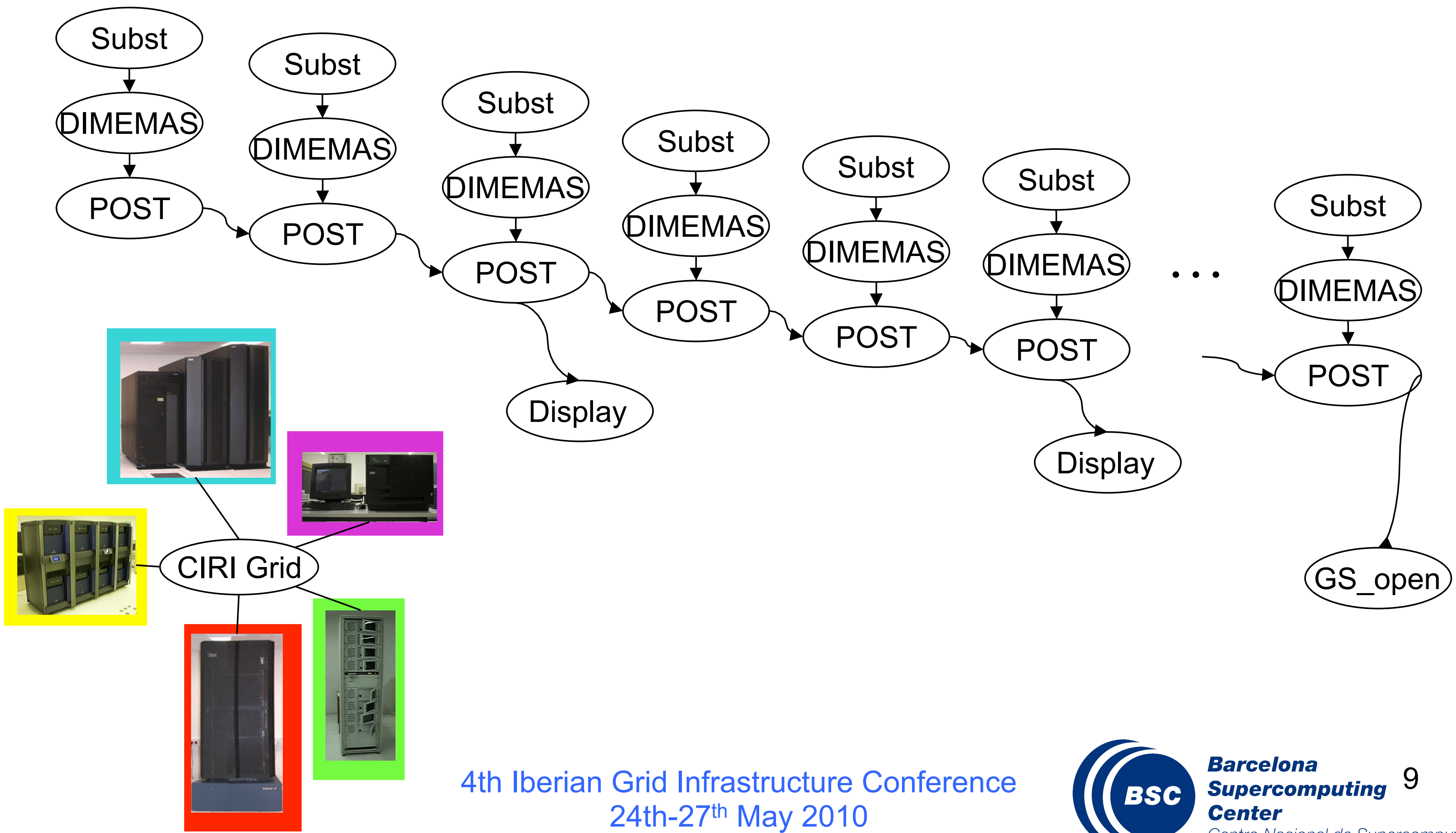
GRID superscalar overview

- Runtime

Input/output data

```
for (int i = 0; i < MAXITER; i++) {
    newBWd = GenerateRandom();
    subst (referenceCFG, newBWd, newCFG);
    dimemas (newCFG, traceFile, DimemasOUT);
    post (newBWd, DimemasOUT, FinalOUT);
    if(i % 3 == 0) Display(FinalOUT);
}
fd = GS_Open(FinalOUT, R);
printf("Results file:\n"); present (fd);
GS_Close(fd);
```

GRID superscalar overview



Parallel task extensions

- Motivation
  - Resources:
    - Limited and heterogeneous resources in grid nodes.
    - Parallel programming models are optimized for exploiting parallelism only for type of resource (shared memory, message passing, grid)
    - Application require big amount of heterogeneous resources are executed in several grid nodes
    - We can not use the same programming model for different resources

Parallel task extensions

- Motivation
  - Applications:
    - Scalability constraints

    (algorithms do not scale for more than X processes)
    - Different levels of parallelism inside the applications (grid, cluster, nodes)
      - Wind power simulation example
        - Simulation for different locations (task parallelism - grid level)
        - For each location simulate different wind directions (“intra-task” parallelism – cluster/node level)

Parallel task extensions

- Motivation
  - Parallel task extensions of GRIDSs goals
    - Hides the platform issues to the user
    - Allows the combination of the different levels of parallelism inside a GRIDSs application.
    - GRIDSs will execute tasks on a capable resource according to the type of parallelism of each task.
- Extensions
  - User interface
  - Runtime
  - Automatic code generation

Parallel task extensions

- User interface
  - New function for defining the parallel description
    - Type of parallelism:
      - Sequential: No parallelism
      - Parallel_sh: several processes in a single host (sh) (openMP, SMPSs, …)
      - Parallel_mh: several processes in multiple hosts (mh) (MPI, UPC,…)
    - Num processes executed by each type of task

Parallel task extensions

- User interface
  - Input arguments of the task can be used to calculate the parallel description parameters.
  - Parallel description parameters are collected at runtime and taken into account for further task management (scheduling, data transfers,…)

Parallel task extensions

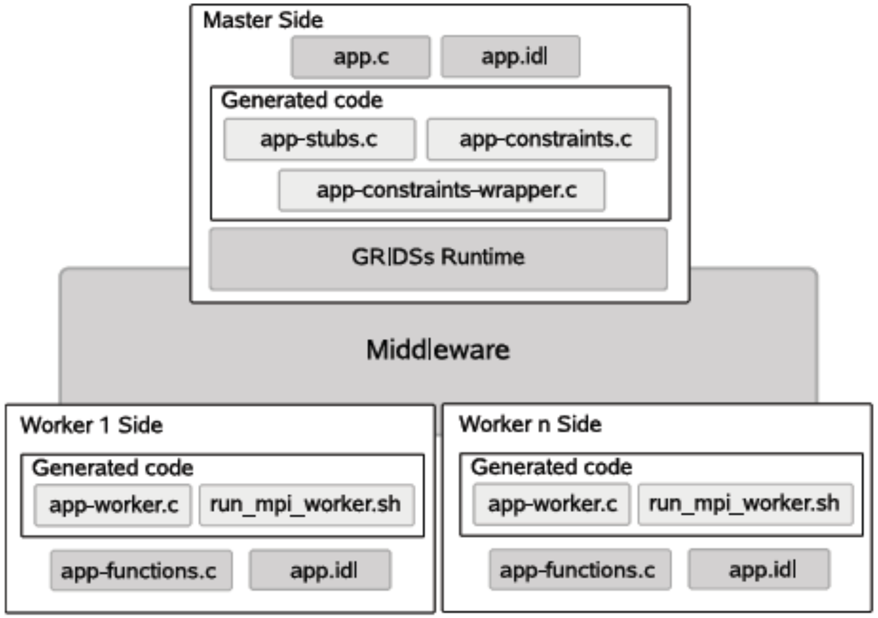
- Runtime extension
  - Scheduling:
    - Selection of multiple slots and hosts per task
    - GRIDSs selects the group of hosts with the smallest cost for each task.
    - Cost depends:
      - Computing cost (provided by the user on the cost function)
      - the number of data transferred on the selected hosts (data locality aware policy)
    - GRIDSs will select the group of hosts where the number of required transfers is the minimum

Parallel task extensions

- Runtime extension
  - Data Management
    - Task files transferred to all the assigned hosts.
  - Task execution:
    - Environment variables required for executing the parallel task.
      - Machine list assigned to a task (GS_MACHINELIST)
      - Number of slots assigned to a task (GS_TASK_NCPUS)

Parallel task extensions

- Automatic code generation
  - Generates new description functions
  - Generates new execution scripts
  - Allows the backward compatibility (generated default values are the same as sequential)



Usage example

```
interface app {
   void simulation_SH(in File input_file, out File output);
   void simulation_MH(in File input_file, in int directions, out File output);
};
```

- Single host

```
void simulation_SH_description(char* input_file, int *numCPUs, char **jobType){
     *jobType = "parallel_sh";
     *numCPUs = 4;
}
```

```
#include "omp.h"
#include <GS_worker.h>

void Simulation_SH(char * input_file, char * output_file){
        float result[200];
        int thr =getenv("GS_TASK_NCPUS");
        omp_set_num_threads(thr);
        #pragma omp parallel
        {
             int ID = omp_get_thread_num();
             sim(inputfile, ID, result);
        }
        writeOutput(output_file, result);
}
```

Usage example

```
interface app {
   void simulation_SH(in File input_file, out File output);
   void simulation_MH(in File input_file, in int directions, out File output);
};
```

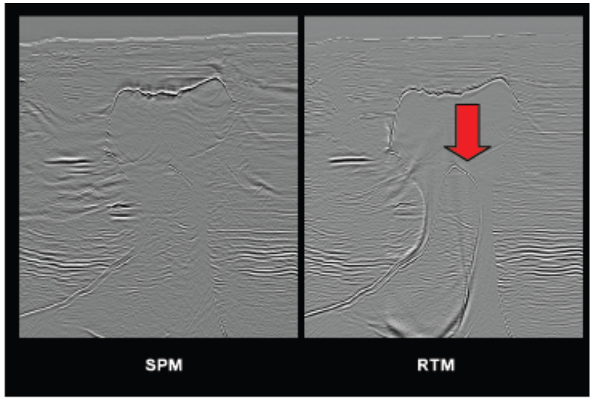
- Multiple host

```
void simulation_MH_description( char* input_file, int wind_dirs, int *numCPUs, char **jobType){
     if (wind_dirs == 1 ){
        *jobType = "sequential";
        *numCPUs = 1;
     }else{
        *jobType = "parallel_mh";
        *numCPUs = wind_dirs;
     }
 }
```

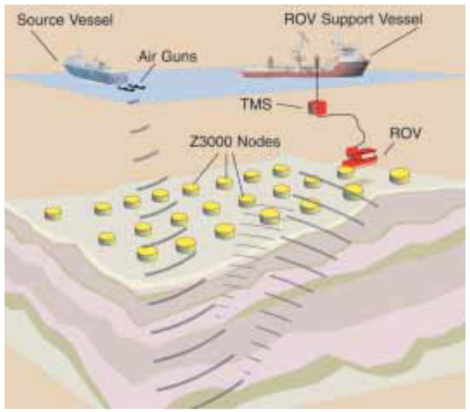
```
void simulation_MH(char* input_file, int wind
        char aux[200];
        int gs_result;
        sprintf(aux,"./run_mpi_worker.sh %s %
        gs_result=GS_System(aux);
}
```

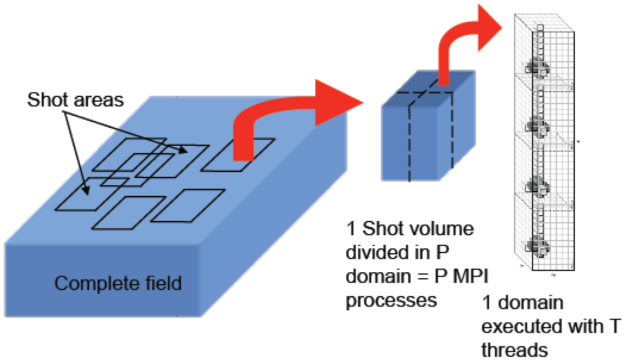
```
#!/bin/bash
# Building the machinefile
touch hostfile.$GS_TASKNUM.$GS_MASTERPID
for node in $GS_MACHINELIST
do
        cat >> hostfile.$GS_TASKNUM.$GS_MASTERPID << EOT
${node}
EOT
done

# Running the simulation
mpirun -np $GS_TASK_NCPUS -machinefile hostfile.$GS_TASKNUM.$GS_MASTERPID \
 $GS_WORKER_BINDIR/simulationMPI $@
result=$?;

# Removing the machinefile
rm -rf hostfile.$GS_TASKNUM.$GS_MASTERPID
exit $result;
```

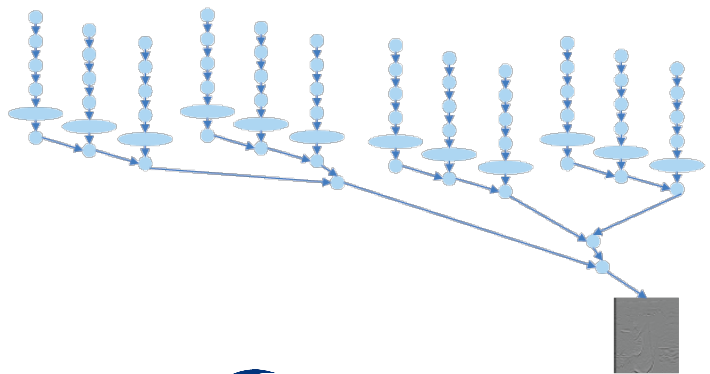
Usage example



- Kaleidoscope project
  - RTM produces proper sub-salt images, computational intensive
  - 1 GRIDSs application per image.
    One task per shot (Grid level)
    (350,000-500,000 tasks/image)
  - Domain decomposition: each task (shot) computed in different nodes (MPI) (cluster level-*parallel_mh*)
  - Domain executed with different threads. (node level-*parallel-sh*)







Conclusions

- GRIDSs extension for supporting parallel tasks
  - Combination of different levels of parallelism
  - Extensions:
    - User interface : description function.
    - Runtime: Scheduling, data management, task execution.
    - Automatic Code Generation: new code generation.
  - Usage examples:
    - Programming complexity is almost the same as programming with the selected parallel programming model.

More information

- GRID superscalar home page:
  www.bsc.es/grid/grid_superscalar