Sample Aplications

This section presents a sample program with GRID superscalar. The program performs a simple optimization search. The application uses as solver the MPI simulator DIMEMAS, which predicts for a given architecture model the behavior of an MPI application. The search space is defined by two variables: L, the latency of the interconnection network of the architecture modeled, and BW, the bandwidth of the same interconnection network. If the goal defined by the user is reached, the DIMEMAS task that detects this throws exception and the execution of the program stops. Figure 1 shows the IDL of this example.

```
#ifndef _OPTIM_IDL
#define _OPTIM_IDL


interface OPT {

void Filter(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 Extract(in File cfgFile, in File DimemasOUT, inout File resultFile);

};

#endif // _OPTIM_IDL
```

Figure 1: IDL for the optimization example

The code is composed of two files: the main program file and the functions file. Figure 2 shows partially the main program file, and figure 3 shows an example of function. GS_Speculative_End is a GRID Superscalar primitive that indicates the end of a code area where an exception can be thrown.

```
#include
#include "optim.h"
....

int main(){
...
GS_On();
j=0;
ini_rand();
while (j< MAX_ITERS){
   getRanges(Lini, BWini, &Lmin, &Lmax, &BWmin, &BWmax);
   for (i=0; i < ITERS; i++){
      L[i] = gen_rand(Lmin, Lmax);
      BW[i] = gen_rand(BWmin, BWmax);

      printf("Latency = %f, BW = %f\n", L[i], BW[i]);
      Filter("nsend.cfg", L[i], BW[i], "tmp.cfg");
      Dimemas("tmp.cfg", "nsend_rec_nosm.trf", Elapsed_goal, "dim_out.txt");
      Extract("tmp.cfg", "dim_out.txt", "final_result.txt");
   }

   getNewIniRange("final_result.txt",&Lini, &BWini);
   j++;
   printf("Iter = %d, Lini = %f, BWini %f\n", j, Lini, BWini);
}
GS_Speculative_End();
GS_Off(0);
}
```

Figure 2: Main program of a sample application

GS_Throw is a GRID Superscalar primitive that throws an exception that will be handled by the GRID Superscalar run-time. The tasks that sequentially appears before a GS_Speculative_End will be executed (some of them concurrently and probable not in the same order of appearance). If a GS_Throw is launched from a given task, all tasks sequentially written after this, will be canceled or undone, depending whether on they are still pending or already finished. The execution will continue normally with the code after the GS_Expeculative_End.

```
void Dimemas(char * cfgFile, char * traceFile, double goal, char * DimemasOUT)
{
char aux[200];
FILE *fp;
double elapsed;
double dist;


putenv("DIMEMAS_HOME=/aplic/DIMEMAS");
sprintf(aux, "TMPDIR=/home/sc03");
putenv(aux);
sprintf(aux, "/aplic/DIMEMAS/bin/Dimemas -o %s %s", DimemasOUT, cfgFile);
gs_result = GS_System(aux);

fp = fopen(DimemasOUT,"r");
fscanf(fp,"%*s %*s %lf", &elapsed);
fclose(fp);

dist = elapsed-goal;
if (dist < 0.0)
      dist = -dist;
if (dist < goal*0.1)
      GS_Throw;

}
```

Figure 3: Example of user function

The complete code for this example is provided below in "Optimization example" link.

Other examples

- C/C++

  Optimization example
  NAS Grid Benchmarks implemented with GRID superscalar
  Matrix multiply

- Java

  Matrix multiply
  Mean value

The samples provided in this section are not ready for execution, since require the GRID superscalar distribution. Also, for the NAS Grid Benchmarks example, the FORTRAN binaries of the NAS Parallel Benchmarks are not provided. In case you are interested in the previous code, please contact: grid-superscalar.at.bsc.es