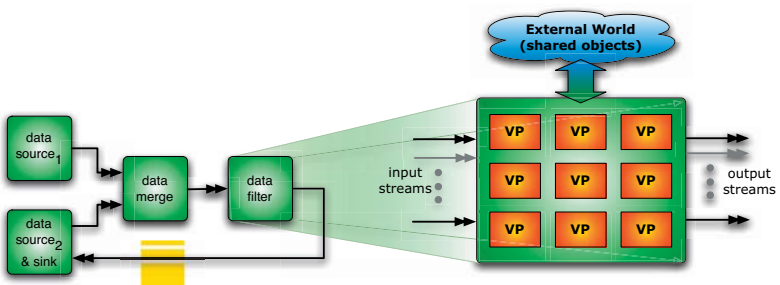
ASSIST

A Software development System based on Integrated Skeleton Technology

ASSIST is a programming environment aimed at providing parallel programmers with user-friendly, efficient, portable, fast ways of implementing parallel applications. It includes a skeleton based parallel programming language (ASSISTcl, cl stands for coordination language) and a set of compiling tools and run time libraries. The ensemble allows parallel programs written using ASSISTcl to be seamlessly run on top of workstation networks supporting POSIX and ACE (the Adaptive Communication Environment, an external, open source library, available from http://www.cs.wust.edu/~schmidt/ACE.html). The programming environment and the coordination language have been designed in a joint project ASI/CNR (Italian National Space Agency and Italian National Research Council), by people of the Dept. of Computer Science of Pisa. Recently, this program terminated and the development of ASSIST has been moved to other italian national research projects (Strategic projects "Legge 449/97" No. 02-00470-ST97 and 02-00640-ST97 and a FIRB project No. RNBNE01KNFP "GRID.it").

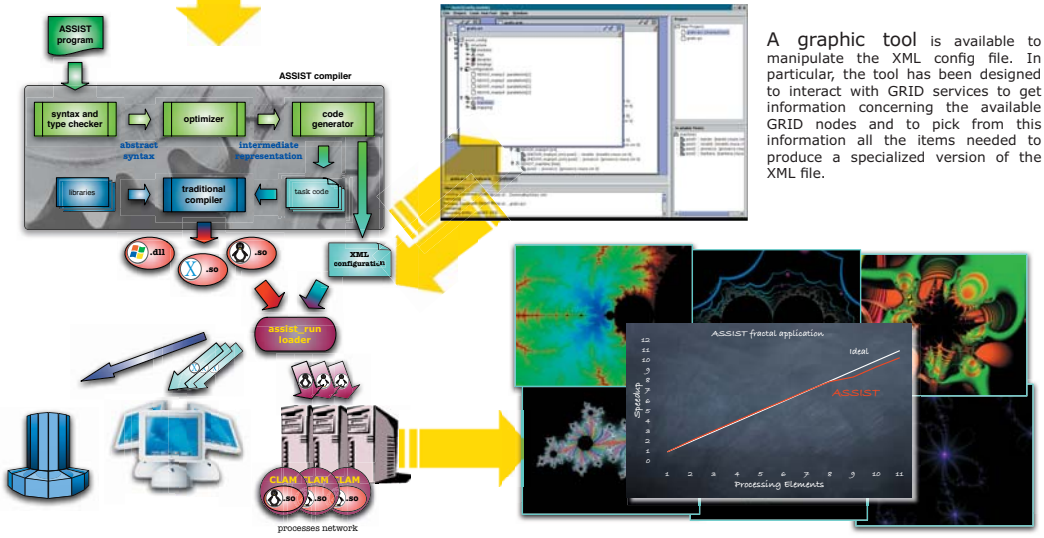
Using ASSISTcl the programmer may structure parallel application as generic graphs of either sequential processes or parallel modules. Nodes in the graphs (i.e. the processes or the parallel modules) are connected by means of data streams. Non-deterministic control is provided to accept inputs from different streams and explicit commands are provided to output items on the output streams. Sequential portions of code can be written using C, C++ or FORTRAN77.



The parmod skeleton allows to define a set of Virtual Processors, to assign tasks (to all of them, or one task per Virtual processor, or one task per partition of Virtual processors), to handle concurrent accesses to state variables, to manage zero or more input stream and zero or more output streams, and to interact with external world accessing (possibly shared) objects via standard protocols (e.g. CORBA).

```
#define N 20
#define MAX_ITER 10

generic main() {
  stream long[N][N] A;
  stream long[N][N] B;

  generate  (output_stream A);
  compute   (input_stream A output_stream B);
  print     (input_stream B);
}

generate(output_stream long A[N][N]) {
  Fgenera (output_stream A);
}

parmod compute (input_stream long A[N][N]
                output_stream long B[N][N]) {
  topology array [i:N][j:N] Pv;
  attribute long S[N][N] scatter S[][] onto Pv[][];
  stream long ris;
  do input_section {
    guard1: on , , A {
      distribution A[*u][*w] scatter to Pv[u][w];
    }
  } while (true)

  virtual_processors {
    elab (in guard1 out ris) {
      VP i,j  {
        for (h=0;h<N;h++) {
          Felab (in S[i][h],S[h][j], S[i][j]
                 out S[i][j]);
        };
        assist_out(ris,S[i][j]);
      }
    }
  }
  output_section {
    output(ris) {
      collects ris from ALL Pv[i][j] ;
      return B;
    }
  }
}

print  (input_stream long B[N][N])
  path<"/usr/include/g++-3">
inc<"fstream">
$c++{
  int i;
  ofstream f;
  f.open ("/tmp/bin/risults.txt");
  for (i=0; i<N; i++) {
    for (int j=0;j<N;j++) {
      f << B[i][j] << " ";
      cerr << B[i][j] << " ";
    }
    cerr << endl;
    f << endl;
  }
  f.close();
}c++$

proc Fgenera (output_stream long A[N][N])
  path<"/usr/include/g++-3">
inc<"iostream","fstream">
$c++{
  static long tmp_A[N][N];
  ifstream f("/tmp/bin/inspt");
  int i,j;
  for(i=0;i<N;++i) {
    for (j=0;j<N;j++) {
      f >> tmp_A[i][j];
      cerr << tmp_A[i][j] << " ";
    }
    cerr << endl;
  }
  assist_out (A, tmp_A);
}c++$

proc Felab (in long a, long b, long s  out long S)
  path<"/usr/include/g++-3">
inc<"iostream">
$c++{
  long t = a+b;
  if (t < s )
    S = t;
  else
    S = s;
}c++$
```

The compiler works on three basic steps: first syntax is parsed and an abstract syntax form is produced. Then a task code (architecture independent parallel abstract code) is produced out of the abstract syntax tree. In this step optimizations are performed aimed at improving program performance and efficiency. Last, POSIX/ACE object code is generated out of the task code, which is suitable to be run onto the ASSIST CLAM (Coordination Language Abstract Machine). The object code is actually produced using standard C++ compilers. Along with the object code, an XML configuration file is generated, holding all the information needed to run the parallel code onto a CLAM configuration. Such information includes parallelism degree, mapping of specialized code to processing nodes and the alike.



An ASSISTcl program follows a syntax which is in part borrowed from C and Pascal. The program has different sections: the first section describes the processes parallel activity graph, the second is used to hold the code describing each process activity.
Parmod (the parallel module skeleton) is used to model most of the parallel activities appearing in ASSISTcl program. A parmod can be specialized to behave as the most common parallelism exploitation pattern (or skeletons, or design patterns). Therefore parmods can be used to express farms, pipelines as well as geometric and data parallel computation patterns.

A graphic tool is available to manipulate the XML config file. In particular, the tool has been designed to interact with GRID services to get information concerning the available GRID nodes and to pick from this information all the items needed to produce a specialized version of the XML file.

ASSIST features

Programmability

Skeleton and coordination technology are exploited in the ASSIST environment in such a way the programmer is not required to handle most of the error prone details he is usually concerned with (processes and communications setup, scheduling, mapping, etc.). The skeletons included in ASSISTcl are far more powerful that the traditional ones. The programmer can therefore implement parallel applications with complex parallelism exploitation patterns.

Code reuse

Sequential portions of code embedded in ASSISTcl programs can be written in C, C++ and FORTRAN, thus enhancing the possibility to reuse existing code.

Rapid prototyping

Programmers can experiment different parallelization strategies just changing a few lines of code and recompiling.

Interoperability

ASSISTcl programs can access external objects via CORBA. A whole ASSISTcl program can be automatically exported (i.e. standard IDL can be automatically generated and proper skeleton code is generated) as a CORBA object to the external world. Furthermore, facilities are present in the language that allow to use external libraries from within the sequential portions of code in the ASSISTcl programs.

Portability

ASSIST has been currently implemented on POSIX/ACE Linux workstation networks. We are currently completing the compiler part needed to target heterogeneous architectures. The POSIX calls used and ACE are available on most modern operating system including different Windows flavours, MacOS X, Linux, BSD, etc.

Performance

ASSISTcl benchmarks demonstrated good (close to ideal) scalability and efficiency on Linux clusters. Efficiency close to 99% has been achieved using medium to coarse grain parallel code.

The ASSIST team

ASSIST has been mostly designed by people of the Dept. of Computer Science, University of Pisa, Italy. Among the others, the following people contributed in either ASSIST(cl) design or implementation: M. Aldinucci, S. Campa, P. Ciullo, M. Coppola, M. Danelutto (project leader), D. Guerri, D. Laforenza, M. Lettere, S. Magini, S. Orlando, A. Paternesi, R. Perego, P. Pesciullesi, A. Petroccelli, E. Pistoletti, L. Potiti, R. Ravazzolo, M. Torquati, L. Vaglini, P. Vitale, M. Vanneschi (group leader), G. Virdis, C. Zoccolo.