ADL: An Algorithm Definition Language for SmartGridSolve

Michele Guidolin, Alexey Lastovetsky
School of Computer Science and Informatics
University College Dublin
Belfield, Dublin 4, Ireland
{ michele.guidolin, alexey.lastovetsky }@ucd.ie

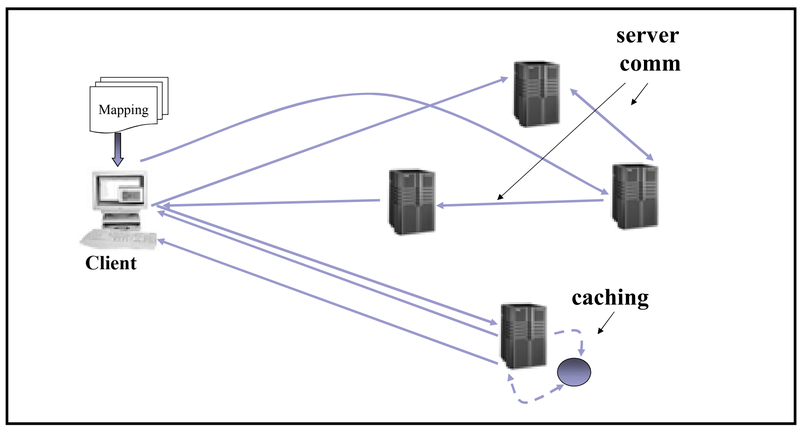
Abstract

SmartGridSolve is an extension of GridSolve that expands the single task map and client-server model of GridRPC by implementing server to server communication and the mapping of a group of tasks. In order to accomplish this functionality SmartGridSolve needs a task graph that highlights tasks' execution order, communication volume and computation volume for a given group of tasks. This work presents the Algorithm Definition Language (ADL), a language that helps the application programmer to easily specify a task graph for any given algorithm. The language is modular, it has a well defined structure and it's syntax is similar to the C language. This poster paper introduces a trivial example of a SmartGridSolve application and the use of ADL to build the relative task graph with an overview of the language syntax.

SmartGridSolve

SmartGridSolve [1], previously known as SmartNetSolve [2], is an extension of GridSolve [3] that supports collective mapping of a group of tasks instead of the original single task map model. In addition the traditional client-server communication model of GridRPC [4] has been extended so that the group of tasks can be collectively mapped on to a network topology which is fully connected. This is a network topology where all servers can communicate directly or the server can cache their outputs locally.

The collective mapping of tasks, with the possibility to use a fully connected network, helps SmartGridSolve find an optimal mapping solution that can exploit fully a Grid environment.



SmartGridSolve, in order to map a group of tasks, needs to build a task graph of the current task calls present in the grid application. SmartGridSolve introduces a new API that automatically generates the task graph. This API works by iterating twice through the application code that contains the task calls to be mapped collectively. On the first iteration of the code each task call is discovered but not executed, then when the last call in the group of tasks is reached the task graph is generated. On the second iteration of the code, after producing the mapping by using the new task graph, the code is normally executed and the task calls are performed.

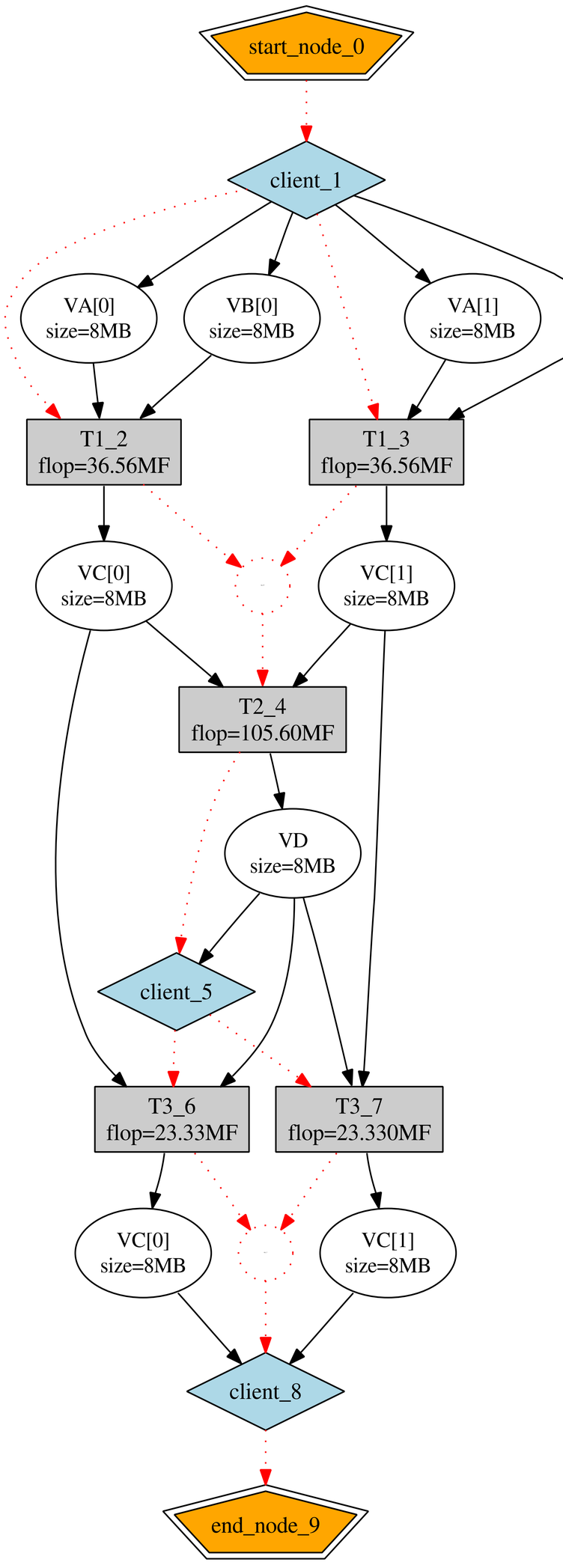
```
...
gs_smart_map("ex_map",auto){
  grpc_call_async(T1_hnd,&id1,VA0,VB0,VC0);
  grpc_call_async(T1_hnd,&id2,VA1,VB1,VC1);
  grpc_wait_all();
  grpc_call(T2_hnd,&id3,VC0,VC1,VD);
  if(F1(VD)<0){
    grpc_call_async(T3_hnd,&id1,VC0,VD,VC0);
    grpc_call_async(T3_hnd,&id2,VC1,VD,VC1);
    grpc_wait_all();
  }
}
...
```

1st Iteration: Discover

2nd Iteration: Execute

One advantage of this method is that the application programmer only has to make minimal modifications to the original GridRPC code. Unfortunately this approach has the restriction that a task graph is not always generated for every kind of algorithm. There are different situations where the automatic task graph generation will not work. A typical example is when, in the code to be mapped, a conditional construct exists that checks a value that cannot be known without executing a remote task call.

The application programmer can choose to create the task graph from a smaller block of code to avoid this problem, but the resulting group of tasks to be mapped will generate a less optimal execution.



Task-Graph

The task graph, a direct acyclic graph (DAG) structure [1], highlights the order of tasks and their synchronisation (whether they are executed in sequence or parallel), the dependencies between tasks, the load of data communication and the task computational volume.

The rectangles in the graph represent remote tasks, the diamonds represent the client computation and the circles represent the data objects. The incoming arrows of these circles indicate their source, whether it is the client or another remote task and the outgoing arrows indicate their destination. The dotted arrows highlight the order of task calls and if the tasks are executed in sequence or parallel. The values inside the circles and rectangles are respectively the size of an object and the computational complexity of a task.

Algorithm Definition Language

The Algorithm Definition Language (ADL), and the respective compiler, allows the application programmer to easily describe all kinds of algorithms for grid applications and generate the corresponding task graph. In the situation where the output of a remote task call can change the flow of execution, the application programmer can know the best way to generate the task graph.

The main goal of ADL is to give a powerful tool to the application programmer that can help him/her to implement a SmartGridSolve application with the best mapping and execution possible.

```
module example (int size, int cond) {
  component:
    task "file.idl"      T1,T2,T3;
  IFO:
    DOUBLE(size)         VA[2],VB[2],VC[2],VD;
  algorithm:
    parallel{
      T1:(VA[0],VB[0]) ->(VC[0]);
      T1:(VA[1],VB[1]) ->(VC[1]);
    }
    T2:(VC[0],VC[1]) ->(VD);
    client:(VD) ->();
    if (cond){
      parfor(int i=0;i<2;i++){
        T3:(VC[i],VD) ->(VC[i]);
      }
    }
}
```

The language syntax is similar to the C language. It uses different modules to define an algorithm and each module, to simplify further the reading, is divided in well defined zones. A zone specifies a characteristic of the algorithm. In ADL we reference an object that is used by a remote task and can be moved anywhere on the Grid as an Identify Flying Object (IFO). The data objects declaration is made in the IFO zone and it is composed of the type, the number of dimensions and the list of IFO names. In the example application the IFOs defined are vectors of double precision numbers and their sizes depend on the value of the parameter "size".

The component zone includes the declaration of the tasks used in the algorithm. The ADL compiler, for each task, requires the number and type of input/output arguments and the eventual computational complexity of the task. All this information can be provided "ad hoc" by the application programmer or retrieved from a gsIDL file [5].

The algorithm zone in the example ADL module describes the flow of execution of the application. A remote task call is composed of two parts, divided by a semicolon. In the first part there is the name of the task called followed by an eventual list of parameters needed. In the second part there is the list of IFOs used as task inputs, followed by an arrow symbol and the list of output IFOs (e.g. VC). This task call syntax is made in a way that easily highlights the parameters passed and the IFOs used as inputs and outputs of a task.

ADL interfaces the main application through the use of gs_smart_map API. The first argument of the API is the same as the previous example. The second argument, instead of the keyword auto, is the keyword ADL. The final arguments match the parameters of the given ADL module.

```
gs_smart_map("ex_map", ADL,"example",size, 1){
  grpc_call_async(T1_hnd,&id1,VA0,VB0,VC0);
  grpc_call_async(T1_hnd,&id2,VA1,VB1,VC1);
  grpc_wait_all();
  grpc_call(T2_hnd,&id3,VC0,VC1,VD);
  if(F1(VD)<0){
    grpc_call_async(T3_hnd,&id1,VC0,VD,VC0);
    grpc_call_async(T3_hnd,&id2,VC1,VD,VC1);
    grpc_wait_all();
  }
}
```

Conclusion

We have presented in this paper the specifications of the ADL language and its compiler. One of the goals of ADL is to overcome the restriction that the automatic task builder exhibits on applications where the flow of execution depends on task call outputs.

We demonstrate that the ADL language overcomes this limitation and permits the application programmer to use SmartGridSolve, with an optimal mapping solution, for any kind of GridRPC application. This work was supported by the Science Foundation Ireland.

[1] T. Brady, M. Guidolin, and A. Lastovetsky. Experiments with SmartGridSolve: Achieving Higher Performance by Improving the GridRPC Model. In Proceedings of the 9th IEEE/ACM Int. Conf. on Grid Computing (Grid 2008), Tsukuba, Japan, 29 Sept. 01 Oct. 2008. IEEE Computer Society.
[2] T. Brady, E. Konstantinov, and A. Lastovetsky. SmartNetSolve: High Level Programming System for High Performance Grid Computing. In Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, 25-29 April 2006. IEEE Computer Society.
[3] A. YarKhan, K. Seymour, K. Sagi, Z. Shi, and J. Dongarra. Recent Developments in GridSolve. International Journal of High Performance Computing Applications, 20(1):131–142, 2006. Sage Science Press.
[4] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In GRID '02: Proceedings of the Third International Workshop on Grid Computing, pages 274–278, London, UK, 2002. Springer-Verlag.
[5] J. Dongarra, K. Seymour, and A. YarKhan. Users' Guide to GridSolve, Version 0.15. University of Tennessee, Knoxville, TN, USA, 2006.