A tiny manual for the GRID superscalar monitor

Rosa M. Badia, Raul Sirvent, Josep M. Perez, Vasilis Dialinos, Pieter Bellens, and Jorge Ejarque

Barcelona Supercomputing Center and UPC, Barcelona, Spain
{rosa.m.badia|pieter.bellens}@bsc.es

Abstract. The GRID superscalar (GS) internally constructs a DAG, referred to as the *task dependency graph* (TDG). The node set of this graph corresponds to the GS tasks, while an edge establishes a true data dependency between the nodes involved. Tasks without unresolved dependencies can be scheduled to run in parallel, hereby producing output data that resolves pending dependencies for other tasks. This cycle essentially repeats itself until all tasks have been computed. The GRID superscalar monitor (GSM) visualizes the (TDG) at runtime, so the user can study the structure of his parallel application and track the progress of execution. The GSM's visualization of the (TDG) will be referred to as the *GSM graph*.

1 What kind of information can I derive from the GSM graph?

Every update of the TDG in the GS runtime is contained in an update to the GSM graph. Conversely, not every update to the GSM graph can be mapped to a single update of GS' TDG. In the current version, there are two *composed events* in the internals of the GS that initiate an update to the GSM graph:

- The addition of a new node to the TDG, **including** all its edges to already existing nodes.
- The state change of a node in the TDG.

The TDG can (for the time being) only grow. There are no provisions to cut branches or hide nodes. The colour of the nodes is an indication of the current state of the task, as well as of the machine that computes the task. Nodes at least start off white, and pass on to grey. These two states represent a state with unresolved dependencies, and a state with all dependencies resolved, but not yet scheduled to run on a machine, respectively. In both cases, there's not an association to a machine yet. The rest of the state sequence depends on the specific version of the GS on your machine. Although it's safe to assume that the state evolves from a light shade, to a darker state, as the task nears the end of its computation. For the current implementation of the GS, only a single chain of states makes sense (figure 1). That is, the state sequence can't be an arbitrary graph, with loops or multi-degree nodes. Tasks are identified by an integer that denotes the order in which these tasks are created by the user program. State,

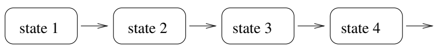

Fig. 1. A directed graph with single-degree nodes and no loops.

task number, and machine (if any) of a task can be found out by hoovering over the node in question. This should open a popup window. Hoovering doesn't mean clicking. As a matter of fact, there shouldn't be any clicking at all in the portion of the GSM window that depicts the GSM graph. At least I can't remember any functionality associated with this action. The actual popping up of the popup window may lag as compared to the hoovering. Whenever the GSM updates the GSM graph, it closes open popup windows. This statement holds for the entire application: every time the GSM updates the GSM graph, functionality is temporarily disabled.

2 GSM implementation.

The GSM is implemented using UDrawGraph (UDG), an interactive graph visualization package from the University of Bremen (`http://www.informatik.uni-bremen.de/uDrawGraph/en/uDrawGraph/uDrawGraph.html`). Just as the GS, the GSM assumes that the Grid consists of a master machine, and worker machines. Additionally, for monitoring purposes, we identify another machine, which doesn't belong to either of the aforementioned groups, the *monitoring machine*. By design, the GSM should be run in the monitoring machine, as not to disturb or influence the Grid computation. Although this is not mandatory: the GSM can also be located on the master or on one of the worker machines, if desired. The GSM sets up an SSH-tunnel between the master machine, and the monitoring machine. Given that the master machine and the monitoring machine or not one and the same. SSH-tunneling to and from the same machine would be just plain silly. Via that SSH-tunnel (and if not, directly), the GS master library (as defined in the GS User's Manual) sends over updates to the GSM. The GSM buffers and processes these messages, and paints the GSM graph via UDG.

3 GSM window

The GSM window consists of a GSM graph area, and a control area. The former has been adequately described in section 1, the latter, being fairly intuitive, is dealt with here. The control area is located in the extreme left part of the GSM window. There's an exit-button (`X`). Clicking the double-bar-button once pauses the generation of the GSM graph, clicking it again resumes the generation. The buttons labeled `AUTO`-something function in the same way. The first click starts the automatic graph improvement feature, resp. the automatic resize feature, the second click turns it off again. Beware of the automatic graph improvement feature. When generating big GSM graphs, this feature may severely slow down the GSM. Another useful button is the magnifying-glass-button, which allows you to scale the graph to your likings. The function of the rest of the buttons should be obvious.

4 Installation and use

- Unpack and unzip the GSM package in the monitoring machine.

  ```
  pbellens@dragcity:/tmp$ tar xzvf monitor_v2.0.tar.gz
  ```

- The directory tree looks a-something like this:

  ```
  ./uDrawGraph-3.1
  ./uDrawGraph-3.1/bin
  ./uDrawGraph-3.1/lib
  ./uDrawGraph-3.1/lib/uDrawGraph
  ./uDrawGraph-3.1/lib/tcl8.4
  ./uDrawGraph-3.1/lib/tcl8.4/http2.5
  ./uDrawGraph-3.1/lib/tcl8.4/http1.0
  ./uDrawGraph-3.1/lib/tcl8.4/opt0.4
  ./uDrawGraph-3.1/lib/tcl8.4/encoding
  ./uDrawGraph-3.1/lib/tcl8.4/msgcat1.3
  ./uDrawGraph-3.1/lib/tcl8.4/tcltest2.2
  ./uDrawGraph-3.1/lib/tk8.4
  ./uDrawGraph-3.1/lib/tk8.4/images
  ./uDrawGraph-3.1/lib/tk8.4/msgs
  ./uDrawGraph-3.1/lib/tk8.4/demos
  ./uDrawGraph-3.1/lib/tk8.4/demos/images
  ./uDrawGraph-3.1/lib/BWidget-1.7.0
  ./uDrawGraph-3.1/lib/BWidget-1.7.0/demo
  ./uDrawGraph-3.1/lib/BWidget-1.7.0/lang
  ./uDrawGraph-3.1/lib/BWidget-1.7.0/BWman
  ./uDrawGraph-3.1/lib/BWidget-1.7.0/tests
  ./uDrawGraph-3.1/lib/BWidget-1.7.0/images
  ./uDrawGraph-3.1/lib/Img1.3
  ./uDrawGraph-3.1/docs
  ./uDrawGraph-3.1/icons
  ./uDrawGraph-3.1/samples
  ./uDrawGraph-3.1/samples/api
  ./uDrawGraph-3.1/samples/graphs
  ./uDrawGraph-3.1/samples/simpleclient
  ./uDrawGraph-3.1/Grid_Superscalar_monitor         <---------------------
  ./uDrawGraph-3.1/Grid_Superscalar_monitor/bitmaps
  ./uDrawGraph-3.1/stuff_for_your_gs_libs
  ```

  We see some graphical libraries, some UDG samples, our patches for our GS master library, and finally (marked by the arrow above), the GSM directory.

- Compile the GSM in the `Grid_Superscalar_monitor`-directory.

  ```
  pbellens@dragcity:Grid_Superscalar_monitor/$ make
  ```

To start visualizing the TDG, **before** running your gridified application, do the following:

- Add the `UDG_HOME` environment variable to your environment with the value of the `uDrawGraph-3.1`-directory.
- Go to the `Grid_Superscalar_monitor`-directory, and start up the monitor:

  ```
  pbellens@dragcity:Grid_Superscalar_monitor$./gs_monitor_start.sh khafre.es
  ```

  If your GS master library happens to be on another machine than `Khafre`, try substituting `khafre.es` with the correct address. More advanced arguments to the `gs_monitor_start`-script are discussed in section 5.
- Start up your GS application in the master machine. The GSM wakes up at the arrival of the first update from the GS master library.

5 More options for the monitor shellscript

```
usage: ./gs_monitor_start.sh [-l local_port] [-r remote_port] machine_name
```

`-r remoteport` specifies the remote port in the master machine whose traffic gets forwarded to the local port in the monitoring machine
`-l localport` specifies the local port in the monitoring machine that accepts the traffic from the master machine

Check your GS distribution to check which ports it uses. If you wish to launch the GSM in the master machine (i.e master machine equals monitoring machine), use `localhost` as `machine_name`. If you wish to connect to the master machine using another username than your current one, replace `machine_name`) with `another_username@machine_name`.

6 Troubleshooting

Although the GSM code is pretty immature, it should be stable enough for normal use (i.e. use that is covered by the description in this document). There's one exception however. If the GS exits with an error, and the shutdown procedure breaks off, sometimes some orphan processess have to be killed manually. If this is the case, you can do the following to restart the GSM with a clean slate:

- In the master machine, locate all processes that are trying to `cat` a file, as well as all the SSH-connections to your monitoring machine. Kill them all.
- In the worker machine, locate all uDrawGraph-processes and all `Graph_Vis_Agent`-processes. Kill them all.

After cleaning up this mess, you shouldn't experience any more problems restarting the GSM. Agreed, this hardly qualifies as high-quality coding. Although it really is an artefact of our use of an SSH-tunnel, and there doesn't seem to be a really clean way to deal with this problem.