PGPVM
Performance Visualization Support for PVM*

Brad Topol Vaidy Sunderam Anders Alund

February 6, 1995

1 Introduction

PGPVM is an enhancement package for PVM 3.3 that produces trace files for use with standard ParaGraph. PGPVM attempts to give an accurate portrayal of applications by minimizing the perturbation inherent with this type of monitoring. PGPVM does not utlize standard PVM tracing but instead its own buffered tracing techniques to provide more accurate monitoring information. Further, PGPVM provides a shellscript that performs some postprocessing and produces a `<file>.trf`, i.e. a standard ParaGraph trace event file. Tachyon removal and clock synchronization are performed during postprocessing when necessary.

1.1 What Must Be Done to Correctly Compile the PGPVM Library

Before the PGPVM library can be correctly compile, please make sure of the following:

I You have installed pvm 3.3 !

II PGPVM expects that it has been placed as a subdirectory of pvm3. If this is not the case, everything should work fine except the fortran examples.

III You are in the pgpvm directory that you must create and have untarred pgpvm.tar. To untar, type `tar -xvf pgpvm.tar`. Further, the $PVM_ROOT has been set and the standard PVM aimk exists and is found by your PATH !

IV You are able to write to $PVM_ROOT/include and to $PVM_ROOT/lib as PGPVM needs to install itself !

*This research was supported in part by the U. S. Department of Energy grant DE-FG05-91ER25105, the National Science Foundation grant NSF ASC-9214149, and the Office of Naval Research grant N00014-93-1-0278.

1.2 What Must Be Done to Correctly Compile the Example Programs

Before the examples can be correctly compiled, please make sure of the following:

I You are in the examples subdirectory of pgpvm and the $PVM_ROOT has been set accordingly and that the standard PVM aimk exists, and can be found from your PATH !

II The fortran examples assume that you have placed pgpvm as a subdirectory of pvm3. If this is not true, you must edit the includes in master1.f and slave1.f. Please note there are two includes in the slave1.f example.

III The `C` examples should work fine no matter where you place the directory as long as #I above is satisfied.

2 Using PGPVM with C programs

Before PGPVM can be utilized, you must compile it. Please refer to the Readme file !

Using PGPVM requires only two minor modifications to a PVM application. First, the application must include a new header file, "pgpvm.h". This should go directly under the standard "pvm3.h" header file. The new header file provides macros that replace normal PVM routines with calls to the PGPVM library. Calls to the PVM library in the application source code need not be modified. The source does however need to be recompiled. Compilation details are addressed in a following section.

The only other modification is the addition of the `pg_tids (int *tid,int nprocs)` library routine to the source code. The array of tid's (nprocs denotes the number of elements in the array) informs PGPVM of which processes should produce tracing information. All processes that wish to participate in tracing must call `pg_tids()` and all must pass it the same tids array that all other processes pass it. Although this sounds difficult, this is usually straightforward as most PVM applications multicast the tids array so that processes can can send messages to each other. Figure 1 illustrates a skeleton PVM application with the necessary source code modifications. It is worth noting that Figure 1 depicts the case where all nodes participate in tracing. Hence all nodes call `pg_tids()` and pass in the full tids array. This does not have to be the case, however. Suppose node 0 does not want to participate in the tracing. Then this node should refrain from calling `pg_tids()` and all other nodes should call `pg_tids()` and pass in a subset of `int *tids`, e.g. `pg_tids(&tids[1],nprocs-1)`.

Once the application has been modified as described and recompiled; upon execution PGPVM produces a single tracefile named `pgfile.<uid>` in the `/tmp` directory. This tracefile can be found on the host machine of the zero'th element of the tids array passed into the `pg_tids()` routine. This file should be moved to the pgpvm/$ARCH directory. From here the user types, `PGSORT pgfile.<uid>`. This shellscript converts the file to standard ParaGraph trace file format and the trace file will appear in the directory as `pgfile.<uid>.trf`. An advanced feature described below allows users to dictate the pathname and filename of where the tracefile is created, and many find this more convenient than retrieving the file from `/tmp`.

```
#include "pvm3.h"
#include "pgpvm.h"  /*first modification discussed*/
main()
{
   ...
   /* declarations, initialize numprocs to be number of processes */
   ...
    mytid = pvm_mytid();
if (pvm_parent()==PvmNoParent) {
tids = (int *) malloc(numprocs * sizeof(int));
tids[0] = mytid;
me = 0;
pvm_spawn("pvmprog",(char **)0,0,0,numprocs-1, &tids[1]);
/* broadcast problem info to all nodes, broadcast tids to all*/
/*nodes                                                       */
pvm_initsend(0);
pvm_pkint(&size,1,1);
pvm_pkint(&dimension,1,1);
pvm_pkint(&numprocs,1,1);
pvm_pkint(tids,numprocs,1);
pvm_mcast(&tids[1],numprocs-1,BROADCAST);

pg_tids(tids,numprocs) /* second source modification discussed*/
}
else { /* Otherwise, receive data from node 0 */
pvm_recv(-1,BROADCAST)
pvm_upkint(&size,1,1);
pvm_upkint(&dimension,1,1);
pvm_upkint(&numprocs,1,1);
tids = (int *) malloc(numprocs * sizeof(int));
pvm_upkint(tids,numprocs,1);

pg_tids(tids,numprocs) /* second source modification discussed*/
/* remember, all nodes that wish to partake in tracing must   */
/* make this call, that is why we see it again      */
    }
    do_work() /* do whatever the application does, etc.. */
}
```

Figure 1: A skeleton PVM application modified in order to utilize PGPVM.

2.1 PGPVM– C Compilation details

As previously touched upon, application programs must be recompiled and linked to the newly created PGPVM library, libpgpvm.a. Moreover, the program must link to the libpvm3.a library. PGPVM does not rely on the PVM group library, libgppvm.a, and this need not be linked unless the application uses the group routines. Further, the application must `#define USE_PGTRACE`. This must be done before the header file `#include "pgpvm.h"` so place this macro in the source code or set it during compilation. Please refer to the Makefile.aimk in the examples subdirectory for proper compilation techniques.

When an application has no need for PGPVM's tracing, simply remove `USE_PGTRACE` macro (either from source or the Makefile, wherever you placed it) and recompile. The other modifications do not have to be removed and the application will execute as if PGPVM routines had not been linked to the application.

3 Using PGPVM with Fortran programs

Before PGPVM can be utilized, you must compile it. Please refer to the Readme file !

Using PGPVM with PVM Fortran programs is only slightly more tedious. Although the user does not add an extra header file nor a `#define USE_PGTRACE` as is the case with C programs, it is necessary that the user modify the names of certain PVM routines. For example, the user must change all calls from `pvmfmytid()` to `pgfmytid()`. Figure 2 lists the mandatory name modifications for PVM Fortran applications. The only other modification

Old Routine Name	New Routine Name
pvmfmytid()	pgfmytid()
pvmfexit()	pgfexit()
pvmfrecv()	pgfrecv()
pvmfprecv()	pgfprecv()
pvmnrecv()	pgfnrecv()
pvmtrecv()	pgftrecv()
pvmfmcast()	pgfmcast()
pvmfsend()	pgfsend()
pvmfpsend()	pgfpsend()

Figure 2: Mandatory name modifications for Fortran Applications

is the addition of the `pgftids(int tids(nprocs),nprocs)` library routine to the source code. The array of tid's (nprocs denotes the number of elements in the array) informs PGPVM of which processes should produce tracing information. All processes that wish to participate in tracing must call `pgftids()` and all must pass it the same tids array that everyone else passes it. Although this sounds difficult, this is usually straightforward as most PVM applications multicast the tids array so that processes can can send messages to each other. Refer to Figure 1 for a `C` example of this or to the Fortran examples, master1.f and slave1.f. It is worth reemphasizing that the tids array passed to `pgftids()` (`pg_tids()` for `C` applications) does not have to contain every tid, only those that wish to partake in producing trace events. In Figure 1 above, all tids partake in tracing since the whole tids array is passed to `pg_tids()` and all nodes call `pg_tids()`. In the Fortran example, master1.f and slave1.f, only slave nodes partake in tracing. We can discern this fact as only the slave nodes call `pgftids()` and they all pass in a tids array that does not contain the master node's tid.

Once the application has been modified as described and recompiled; upon execution PGPVM produces a single tracefile named `pgfile.<uid>` in the `/tmp` directory. This tracefile can be found on the host machine of the zero'th element of the tids array passed into the `pgftids()` routine. This file should be moved to the pgpvm/$ARCH directory. From here the user types, `PGSORT pgfile.<uid>`. This shellscript converts the file to standard ParaGraph trace file format and the trace file will appear in the directory as `pgfile.<uid>.trf`. An advanced feature described below allows users to dictate the pathname and filename of where the tracefile is created, and many find this more convenient than retrieving the file from `/tmp`.

3.1 PGPVM– Fortran Compilation details

As previously touched upon, application programs must be recompiled and linked to the PGPVM library, libpgpvm.a. This is in addition to linking to the standard PVM libraries that Fortran programs link to. Please refer to the Makefile.aimk in the examples subdirectory of PGPVM for proper compilation techniques.

When an application has no need for PGPVM's tracing, simply comment out `pgftids()` and recompile. The other modifications do not have to be removed and the application will execute as if PGPVM routines had not been linked to the application.

4 PGPVM– Advanced Features

Currently, PGPVM has three advanced features, all very easy to use. The first of these, `pg_outfile(char *path)` (`pgfoutfile(string)` for Fortran[1]) allows the user to specify the path of where the tracefile should be created. Only the process who is tid[0] can make this call, and if others call it, they will be ignored. If this leads to an error, e.g. invalid path, no write permissions, etc. PGPVM defaults to creating the tracefile as `pgfile.<uid>` in `/tmp`.

The second advanced feature is the routine `pg_close()` (`pgfclose()` for Fortran). The `pg_close()` routine allows the production of trace events to terminate before `pvm_exit()` is called. If this is used, all nodes should call it simultaneously as it uses a barrier to wait until all processes finish producing trace events. If this call is not used, processes produce trace events up until `pvm_exit()`. This necessarily implies that all nodes that participate in tracing need to eventually call `pg_close()` or `pvm_exit()`.

The third advanced feature of PGPVM is the `pg_chprefix(char *key)` routine (`pgfchprefix(string key)` for Fortran[2]). The `pg_chprefix()` routine was added as

[1] For Fortran, the string must be null terminated to work properly

[2] Again, this string must be null terminated to work properly.

some users did not want PGPVM implementing a barrier at the end of the application. Recall that PGPVM does this to gather and concatenate trace files. **To use this routine, `pg_chprefix()` must be placed directly before the call to `pg_tids()` by all processes that participate in tracing. If this is not followed, `pg_chprefix()` will result in a no-op !** When `pg_chprefix()` is used, PGPVM refrains from gathering and concatenating the trace files and this is now the user's responsibility. Since PGPVM creates unique filenames for the trace files, it can be tedious to write a shellscript to collect the files. The string passed into `pg_chprefix()` is prepended to every trace filename created during this execution of the application. The user may now write a shellscript that is searching various `/tmp` directories for `<key>.*`. As long as you pick a key that you are confident no one else is using (e.g., your mother's maiden name), you should be fine.

Care must be taken when using this routine, it may or may not be worth the headaches it brings ! I say this for a couple of reasons. First of all, when `pg_chprefix()` is **not** used, PGPVM takes care of removing all the individual trace files. Now, you are responsible for removing them, and if ignored they can consume up a large amount of disk space. Further, if you do leave these files around in `/tmp`, the next time you run the application you must make sure to choose another unique key. Not choosing a new key and not deleting old trace files can cause your shellscript to gather old, bogus trace files and this causes severe grief ! Further, you must guarantee that **all nodes that call `pg_tids()` must first call `pg_chprefix()`**. This is because if some processes think they are suppossed to perform the barrier, and others think they are not supposed to perform the barrier, your program will **deadlock** ! Make sure that all processes make this call and they all do it before they call `pg_tids()`

Personally, I hope you don't need to use this routine, however there are some cases when you must, and these are described in the next section. In the examples subdirectory, `testpg2.c` and `chprefix.c` contain examples on how to use this routine.

4.1 When must you use `pg_chprefix()` ?

I can only think of one scenario when your application must use this routine. When PGPVM gathers the trace files, it necessarily uses two message tags, 0x22cb173b and 0x22cb173c. I have attempted to pick tags that most applications don't use, and if your applications don't use these tags you're fine, even if you use wildcards in your PVM receive calls. If your code does use the two message tags that I use, you can either modify your code or modify the pgpvm library to use tags your code doesn't use. If neither of these options are feasible for you, then you must resort to `pg_chprefix()`. Also worth mentioning is that when you use `pg_chprefix()`, please do not pass it NULL for a string.

5 Comments / Questions

Please send comments and questions regarding PGPVM to
Brad Topol
topol@cc.gatech.edu
or

Vaidy Sunderam
vss@mathcs.emory.edu

6 Obtaining ParaGraph

ParaGraph, a performance visualization tool created by Michael T. Heath and Jennifer E. Finger, is available free over the internet. For instructions on how to obtain ParaGraph, simply send an electronic mail message to netlib@ornl.gov containing the text, “send index from paragraph”.