CEE 618 Scientific Parallel Computing (Lecture 8): SSH, Profiling, and Scalapack

Albert S. Kim

Department of Civil and Environmental Engineering
University of Hawai'i at Manoa
2540 Dole Street, Holmes 383, Honolulu, Hawaii 96822

Table of Contents

Outline

Advantages of using subroutines and functions

- A subroutine/function is a portion of code within a larger program that usually does something very specific, and that can be called from anywhere in the program.
- What are advantages of using subroutines/functions?
- Programmers can
  1. decompose a long/complex program into small, simple and independent steps,
  2. reduce code duplication,
  3. reuse subroutines/functions for future develpment,
  4. divide a large programming task among several programmers,
  5. hide unnecessarily detailed information, and
  6. improve traceability for debugging and profiling.

Profiling

- You develop a large set of programs consisting of a number of subroutines and functions, or you download an open-source package of complex programming structures for your own purposes.
- Programs run really slow so that you want to improve the code efficiency in terms of running time.
- How do you know which subroutines are slowest or taking most of running time?
- Profiling allows you to learn where your program spent its time and which functions called which other functions while it was executing.

Profiling using gprof (GNU profiling)

1. In general, we compile codes using "make" utility:
   `$ make`
2. Be sure that "**-pg**" option follows after "ifort" in your Makefile. So, your compiling command should look like:
   `$ ifort -pg mycode.f90 -o mycode.x`
3. Run the compiled executable file: `$ make run`
   which will execute "`mycode.x`". During the execution, "`gmon.out`" will be generated and used for profiling.
4. After the job is done, see the profile where gmon.out exists:
   `$ gprof mycode.x`
   You will see a lot of text message scrolled over.
5. To save the profiling results:
   `$ gprof mycode.x > mygprof.txt`
   Note: if you use "`>>`", you can **append** contents to an existing file.
6. Open `mygprof.txt` to see profiling results.

Review – LAPACK subroutines, dgetrf & dgetrs

```
1  program LUlapack
   implicit none
3  integer          :: i,j, ipiv(3), info
   double precision :: a(3,3)=(/1.,1.,2.,3.,1.,3.,1.,2.,4./)
5  double precision :: b(3)=(/1.,0.,0./)

7  open(11,file='lulapack.dat')
   !                  Display the given matrix, A and b
9  write(11,*)
   do i=1,3
11    write(11,"(4(2x,F12.6)) ") (a(i,j),j=1,3), b(i)
   end do
13 !                  Decomposition of the given matrix A
   call dgetrf(3,3,a,3,ipiv,info)
15 !                  Display the decomposed matrix, A and b
   write(11,*)
17 do i=1,3
      write(11,"(4(2x,F12.6)) ") (a(i,j),j=1,3), b(i)
19 end do
   !                  Solving for x with the decomposed matrix using backsubsitution
21 call dgetrs('N',3,1,a,3,ipiv,b,3,info)
   !                  Display the decomposed matrix, A and the solution x
23 write(11,*)
   do i=1,3
25    write(11,"(4(2x,F12.6)) ") (a(i,j),j=1,3), b(i)
   end do
27 stop
   end
```

codes/profiling/LUlapack–gprof/LU3dlapack.f90

Result: lulapack.dat

```
 2    1.000000    3.000000    1.000000    1.000000
      1.000000    1.000000    2.000000    0.000000
 4    2.000000    3.000000    4.000000    0.000000

 6    2.000000    3.000000    4.000000    1.000000
      0.500000    1.500000   -1.000000    0.000000
 8    0.500000   -0.333333   -0.333333    0.000000

10    2.000000    3.000000    4.000000    2.000000
      0.500000    1.500000   -1.000000    0.000000
12    0.500000   -0.333333   -0.333333   -1.000000
```

codes/profiling/LUlapack–gprof/lulapack.dat

How to

```
[mpiuser@fractal LUlapack-gprof]$ make
ifort -pg  LU3dlapack.f90  /opt/local/lib/liblapack.a /opt/local/lib/libgoto2.a -o LU3dlapack.x
[mpiuser@fractal LUlapack-gprof]$ make run
./LU3dlapack.x
cat lulapack.dat

      1.000000      3.000000      1.000000      1.000000
      1.000000      1.000000      2.000000      0.000000
      2.000000      3.000000      4.000000      0.000000

      2.000000      3.000000      4.000000      1.000000
      0.500000      1.500000     -1.000000      0.000000
      0.500000     -0.333333     -0.333333      0.000000

      2.000000      3.000000      4.000000      2.000000
      0.500000      1.500000     -1.000000      0.000000
      0.500000     -0.333333     -0.333333     -1.000000
[mpiuser@fractal LUlapack-gprof]$ make gprof
gprof LU3dlapack.x  -z > gprof_LU3dlapack.txt
[mpiuser@fractal LUlapack-gprof]$ make que
qsub LU3dlapack.pbs
2785.fractal.eng.hawaii.edu
```

codes/profiling/LUlapack–gprof/how–to–screen.txt

Analysis file:

`codes/profiling/LUlapack-gprof/gprof_LU3dlapack.pdf`

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00      0.12     0.12                             alloc_mmap
  0.00      0.12     0.00        1     0.00     0.00  MAIN__
.
.
.

  0.00      0.12     0.00                             dgetrf_
  0.00      0.12     0.00                             dgetrs_
.
.
.
-----------------------------------------------

                                     <spontaneous>
[123]    0.0    0.00    0.00                 dgetrf_ [123]
-----------------------------------------------

                                     <spontaneous>
[124]    0.0    0.00    0.00                 dgetrs_ [124]
-----------------------------------------------

.
.
.
 [359] __intel_memset         [123] dgetrf_              [255] for__write_args
 [360] __intel_new_proc_init  [124] dgetrs_              [256] for__write_output
```

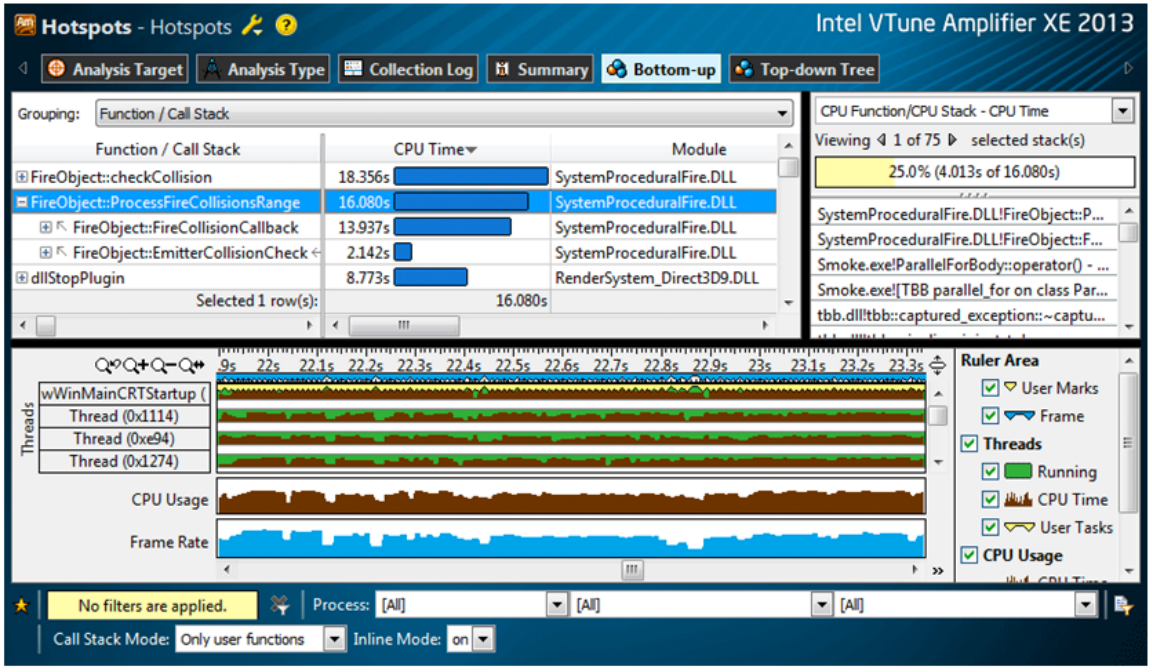
codes/profiling/LUlapack–gprof/gprof_LU3dlapack_part.txt

gprof: useful options

- If you give the '-z' option, gprof will mention all functions in the flat profile, even those that were never called, and that had no time spent in them. This is useful in conjunction with the '-c' option for discovering which routines were never called.
- The '-c' option causes the static call-graph of the program to be discovered by a heuristic which examines the text space of the object file. Static-only parents or children are indicated with call counts of '0'.

`codes/profiling/LUlapack-gprof/gprof_LU3dlapack_graph.pdf`

VTune



Intel VTune Amplifier XE 2013 is the premier profiler for C, C++, C#, Fortran, Assembly and Java.

Makefile

```
lapack=/opt/lapack/liblapack.a
blas=/opt/blas/libgoto2.a
srccode= LU3dlapack.f90
exefile= LU3dlapack.x
gproffile= gprof_LU3dlapack.txt
amplxe=/opt/intel/vtune_amplifier_xe_2013/bin64/amplxe-cl

exec:
	ifort $(prof)  $(srccode)  $(lapack) $(blas) -o $(exefile)

vtune:
	/opt/intel/vtune_amplifier_xe_2013/bin64/amplxe-cl  -collect hotspots ./LU3dlapack.x
	/opt/intel/vtune_amplifier_xe_2013/bin64/amplxe-cl  -report  hotspots ./LU3dlapack.x

edit:
	vim $(srccode)

clean:
	rm -f  *.x *.err *.out *.o
```

codes/profiling/LUlapack–vTune/Makefile

Analysis Output

```
1
[mpiuser@fractal LUlapack-vTune]$ /opt/intel/vtune_amplifier_xe_2013/bin64/amplxe-cl -collect
    hotspots ./LU3dlapack.x
3 Warning: A symbol file is not found. The call stack passing through '/cluster/mpiuser/CEE618
    -2013S/Lecture08-Scala-ssh-prof/gprof/LUlapack-vTune/LU3dlapack.x' module may be
    incorrect.
Using result path '/cluster/mpiuser/CEE618-2013S/Lecture08-Scala-ssh-prof/gprof/LUlapack-vTune/
    r001hs'
5 Executing actions 50 % Generating a report
Summary
7 ----------

9 Elapsed Time: 0.150
CPU Time: 0.130
11 CPU Usage: 0.938
Executing actions 100 % done
13 [mpiuser@fractal LUlapack-vTune]$ /opt/intel/vtune_amplifier_xe_2013/bin64/amplxe-cl -report
    hotspots ./LU3dlapack.x
Using result path '/cluster/mpiuser/CEE618-2013S/Lecture08-Scala-ssh-prof/gprof/LUlapack-vTune/
    r001hs'
15 Executing actions 50 % Generating a report
Function Module CPU Time
17 ----------- ------------- ----------
alloc_mmap LU3dlapack.x 0.130
19 Executing actions 100 % done
```

codes/profiling/LUlapack-vTune/vtune.txt

Outline

Remote Login using ssh: Windows SSH

1. Use program “SSH Secure Client Shell” to login fractal.eng.hawaii.edu.

2. Click “Quick Connect” and input the following information
   1. Host Name: **fractal.eng.hawaii.edu**
   2. User Name: **mpiuserNNN**
   3. Port Number: **22**
   4. Authentication Method: **Password**

Remote Login using PuTTY

1. Watch a tutorial video
   `http://youtu.be/ma6Ln30iP08`
2. Read manual of how to use PuTTY and Filezilla:
   `PuTTY-Filezilla.pdf`
3. After students become familiar to PuTTY, password login will be disabled.
4. Keep your private key file very private.

Outline

Parallel Computing: $\mathbf{A} \cdot \mathbf{b} = \mathbf{c}$

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} & \mathbf{5} \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \\ \mathbf{1} \end{pmatrix} = \begin{pmatrix} 10 \\ 20 \\ 30 \\ 40 \\ \mathbf{50} \\ 60 \\ 70 \\ 80 \\ 90 \\ 100 \end{pmatrix}$$

1. In parallel computation, b **will be shared** by all slave processors,
2. the master will send **one row of** A to a slave processor, and
3. the worker will calculate **a value of** c and return it to the master.

* *Note: this example is to practice MPI routines. Better methods of parallel linear algebra will be taught later.*

Procedure: $\mathbf{A} \cdot \mathbf{b} = \mathbf{c}$

1. The master **broadcasts** b vector.
2. The master sends first 3 (=numprocs -1) rows of A to 3 worker processes in a sequence, i.e., *the first row to processor 1, the second row to the processor 2, and* ***the third row to the processor 3***.
3. Each worker process receives a row of A, calculates $ans = (\text{a row of } A) \cdot b$, and sends ans to the master.
4. The master receives ans as c_i and sends the next row of A to the slave who just returned ans. **First come, first service.**
5. This procedure continues until the master sends **all the rows of A** to slaves in a sequence.
6. If the master sent the last row, it also sends **job-finishing messages** to slaves. Then, each slave **finalizes the MPI job**.

Initial Part 1

```
c**********************************************************************
c     matmul.f - matrix - vector multiply, simple self-scheduling version
c**********************************************************************
      program main

      include 'mpif.h'

      integer MAX_ROWS, MAX_COLS, rows, cols
      parameter (MAX_ROWS = 1000, MAX_COLS = 1000)
      double precision a(MAX_ROWS,MAX_COLS), b(MAX_COLS), c(MAX_COLS)
      double precision buffer(MAX_COLS), ans

      integer myid, master, numprocs, ierr, status(MPI_STATUS_SIZE)
      integer i, j, numsent, numrcvd, sender, job(MAX_ROWS)
      integer rowtype, anstype, donetype

      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
      if (numprocs .lt. 2) then
         print *, "Must have at least 2 processes!"
         call MPI_ABORT( MPI_COMM_WORLD, 1 )
         stop
      endif
      print *, "Process ", myid, " of ", numprocs, " is alive"
```

Initial Part 2

```
      rowtype  = 1
      anstype  = 2
      donetype = 3

      master   = 0
      rows     = 10
      cols     = 10

      if ( myid .eq. master ) then
c        master initializes and then dispatches
c        initialized a and b
         do 20 i = 1,cols
            b(i) = 1
            do 10 j = 1,rows
               a(i,j) = i
 10         continue
 20      continue
      end if

c     broadcast b to each other process
      call MPI_BCAST(b, cols, MPI_DOUBLE_PRECISION, master,
     $     MPI_COMM_WORLD, ierr)
```

Master Part 1

```
      if ( myid .eq. master ) then
         numsent = 0
         numrcvd = 0


c        send a row to each other process
         do 40 i = 1,numprocs-1
            do 30 j = 1,cols
               buffer(j) = a(i,j)
30          continue
            call MPI_SEND(buffer, cols, MPI_DOUBLE_PRECISION, i,
     $           rowtype, MPI_COMM_WORLD, ierr)
            job(i)  = i
            numsent = numsent+1
40       continue
```

Master Part 2

```
      do 70 i = 1,rows
         call MPI_RECV(ans, 1, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,
     $           anstype, MPI_COMM_WORLD, status, ierr)
         sender = status(MPI_SOURCE)
         c(job(sender)) = ans

         if (numsent .lt. rows) then
            do 50 j = 1,cols
               buffer(j) = a(numsent+1,j)
50          continue
            call MPI_SEND(buffer, cols, MPI_DOUBLE_PRECISION, sender,
     $              rowtype, MPI_COMM_WORLD, ierr)
            job(sender) = numsent+1
            numsent     = numsent+1
         else
         call MPI_SEND(1, 1, MPI_INTEGER, sender, donetype,
     $           MPI_COMM_WORLD, ierr)
         endif
70    continue

c     print out the answer
      do 80 i = 1,cols
         print *, "c(", i, ") = ", c(i)
80       continue
```

Slave Part

```
      else

c
 90       call MPI_RECV(buffer, cols, MPI_DOUBLE_PRECISION, master,
     $          MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)

          if (status(MPI_TAG) .eq. donetype) then
             go to 200

          else
             ans = 0.0
             do 100 j = 1,cols
                ans = ans+buffer(j)*b(j)
 100         continue
             call MPI_SEND(ans, 1, MPI_DOUBLE_PRECISION, master, anstype,
     $            MPI_COMM_WORLD, ierr)
             go to 90
          endif

      endif


 200  call MPI_FINALIZE(ierr)
      stop
      end
```

pmm_master.f

```
      numsent = 0
      numrcvd = 0

c     send a row to each other process
      do 40 i = 1,numprocs-1
         do 30 j = 1,cols
            buffer(j) = a(i,j)
30       continue
         call MPI_SEND(buffer, cols, MPI_DOUBLE_PRECISION, i,
     $        rowtype, MPI_COMM_WORLD, ierr)
         job(i)  = i
         numsent = numsent+1
40    continue

      do 70 i = 1,rows
         call MPI_RECV(ans, 1, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,
     $        anstype, MPI_COMM_WORLD, status, ierr)
         sender = status(MPI_SOURCE)
         c(job(sender)) = ans

         if (numsent .lt. rows) then
            do 50 j = 1,cols
               buffer(j) = a(numsent+1,j)
50          continue
            call MPI_SEND(buffer, cols, MPI_DOUBLE_PRECISION, sender,
     $           rowtype, MPI_COMM_WORLD, ierr)
            job(sender) = numsent+1
            numsent     = numsent+1
         else
         call MPI_SEND(1, 1, MPI_INTEGER, sender, donetype,
     $        MPI_COMM_WORLD, ierr)
         endif
70    continue

c     print out the answer
      do 80 i = 1,cols
         print *, "c(", i, ") = ", c(i)
80    continue
```

pmm_slave.f

```
90    call MPI_RECV(buffer, cols, MPI_DOUBLE_PRECISION, master,
     $     MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)

      if (status(MPI_TAG) .eq. donetype) then
         go to 200

      else
         ans = 0.0
         do 100 j = 1,cols
            ans = ans+buffer(j)*b(j)
100      continue
         call MPI_SEND(ans, 1, MPI_DOUBLE_PRECISION, master, anstype,
     $        MPI_COMM_WORLD, ierr)
         go to 90
      endif

      endif
```

Outline

Scalapack

- **Scalable** LAPACK
- redesigned for distributed memory MIMD (Multiple Instruction Multiple Data) **parallel** computers
- written in a Single-Program-Multiple-Data style
- assumes matrices are laid out in a two-dimensional **block cyclic** decomposition.

LAPACK

- Linear Algebra PACKage[1]
- written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.
- Efficient and optimized reorganization of **EISPACK** and **LINPACK**.
- Requires **BLAS**[2] (Basic Linear Algebra Subprograms). Original BLAS does not perform well so that **ATLAS**[3] (Automatically Tuned Linear Algebra Software) is recommended. Currently, the fastest BLAS is **GotoBLAS** from TACC[4] (Texas Advanced Computing Center).
- We are using GotoBLAS.

[1] http://www.netlib.org/lapack/
[2] http://www.netlib.org/blas/
[3] http://www.netlib.org/atlas/
[4] http://www.tacc.utexas.edu/

- EISPACK[5] computes the **eigenvalues** and **eigenvectors** of nine classes of matrices:
  1. complex general,
  2. complex Hermitian,
  3. real general,
  4. real symmetric,
  5. real symmetric banded,
  6. real symmetric tridiagonal,
  7. special real tridiagonal,
  8. generalized real, and g
  9. eneralized real symmetric matices.
- LINPACK[6] analyzes and solves
  1. linear equations and
  2. linear least-squares problems.

[5] http://www.netlib.org/eispack/
[6] http://www.netlib.org/linpack/

Necessary Libraries

1. Serial Libraries
   - BLAS (Basic Linear Algebra Subprograms)
     Alternatives: ATLAS (Automatically Tuned Linear Algebra Software) and GotoBLAS[7]
   - LAPACK (Linear Algebra Package)
2. Parallel Libraries
   - BLACS (Basic Linear Algebra Communication Subprograms)
     - A linear algebra oriented message passing interface.
     - Originally written in C with MPI, linked to FORTRAN, but 32 bit version, very problematic with current 64 bit hardware)
     - Recently, Intel provides 64 bit version of BLACS!
     - Intel Math Kernel Library supports very well.
   - SCALAPACK (Scalable LAPACK)
   - PBLAS (Parallel Basic Linear Algebra Subprograms)[8]
3. MPI (Message Passing Interface): MPICH[9] or OpenMPI[10].

[7] http://www.tacc.utexas.edu/tacc-projects/gotoblas2
[8] http://www.netlib.org/scalapack/pblas_qref.html
[9] http://www.mcs.anl.gov/research/projects/mpich2/
[10] http://www.open-mpi.org/

Memory Requirement

1. One double precision number = 64 bits = 8 bytes
2. 2 Gbytes
   - 2GB x (1 double precision / 8 bytes)
   - 250 millions double precision numbers
   - approximately $(15,810)^2 \approx (16,000)^2$
3. In principle, a processor having 2GB of memory can solve a linear equation of the maximum matrix size as big as about $16,000 \times 16,000$. In reality, smaller than estimated here.
4. This is often limited by a compiler's memory model: up to 2GB, small/medium memory model is efficient and fast. **Allocate!**
5. If one should solve a linear system of, for example, $20,000 \times 20,000$ matrix, then parallel computation is the only method available. Or, a shared memory machine can be use.
6. With 4 available processes, each process should have a $10,000 \times 10,000$ sub-matrix (or called block matrix), requiring memory of $10^8 \times 8 = 800MB = 0.8GB$, (i.e., 100 Million double precision # = 800 MB).

Scalapack

- **Scalable** LAPACK
- redesigned for distributed memory MIMD (Multiple Instruction Multiple Data) **parallel** computers
- written in a Single-Program-Multiple-Data style
- assumes matrices are laid out in a two-dimensional **block cyclic** decomposition.

PDGEMM: part 1

```
1  !  Parallel Matrix Multiplication Example
   !
3  ! A =
   !       1 2 3 4 5 6 7 8
5  !       1 2 3 4 5 6 7 8
   !       1 2 3 4 5 6 7 8
7  !       1 2 3 4 5 6 7 8
   !       1 2 3 4 5 6 7 8
9  !       1 2 3 4 5 6 7 8
   !       1 2 3 4 5 6 7 8
11 !       1 2 3 4 5 6 7 8
   !
13 ! B =
   !       1 1 1 1 1 1 1 1
15 !       2 2 2 2 2 2 2 2
   !       3 3 3 3 3 3 3 3
17 !       4 4 4 4 4 4 4 4
   !       5 5 5 5 5 5 5 5
19 !       6 6 6 6 6 6 6 6
   !       7 7 7 7 7 7 7 7
21 !       8 8 8 8 8 8 8 8
   !
23 ! C = A * B
   !
25 ! A_decom = 8 x 2  for each process
   ! B_decom = 8 x 2  for each process
27 ! C_decom = 8 x 2  for each process
   !
29 ! Process Grid = 1 x 4
   !
31 !
```

codes/scalapack/pmm/pmm.f90.1

PDGEMM: part 2

```
implicit none
      include 'mpif.h'

integer :: my_id, myrow,mycol, num_procs , ictxt , info , rc

integer :: nprow = 1, npcol = 4
integer :: m     = 8, n     = 8, lld    = 8
integer :: mb    = 8, nb    = 2
!
! num_proc = nprow * npcol
! mb = m / nprow
! nb = n / npcol
!
integer :: desca (9), descb (9), descc (9)

integer :: i, j

double precision :: a(8,2), b(8,2) , c(8,2)

CHARACTER*(20) PNAME
INTEGER RESULTLEN

! MPI
call MPI_INIT( info )
call MPI_COMM_RANK( MPI_COMM_WORLD, my_id, info )
call MPI_COMM_SIZE( MPI_COMM_WORLD, num_procs, info )
call MPI_GET_PROCESSOR_NAME(PNAME, RESULTLEN, info)
write(*,*) PNAME, RESULTLEN, info
```

codes/scalapack/pmm/pmm.f90.2

PDGEMM: part 3

descinit[11]

```
1  ! BLACS
   call blacs_pinfo ( my_id, num_procs)
3  call blacs_get (-1, 0, ictxt)
   call blacs_gridinit (ictxt, 'R', nprow, npcol )
5  call blacs_gridinfo (ictxt,  nprow, npcol, myrow, mycol )
   write(*,*) my_id, num_procs, myrow, mycol
7
   ! Matrix DESCription
9  call descinit ( desca, m, n, mb, nb, 0, 0, ictxt, lld, info)
   call descinit ( descb, m, n, mb, nb, 0, 0, ictxt, lld, info)
11 call descinit ( descc, m, n, mb, nb, 0, 0, ictxt, lld, info)

13 ! Matrix initialization

15 do i = 1, mb
     do j = 1, nb
17     a(i,j) = dble(j) + 2*my_id
       b(i,j) = dble(i)
19   enddo
   enddo
21
   call pdgemm ( 'N', 'N',   m,      n,      m, &
23             1.0D0,   a,   1,      1, desca, &
                   b,   1,   1, descb, 0.0D0, &
25                 c,   1,   1, descc)
```

codes/scalapack/pmm/pmm.f90.3

[11] http://www.netlib.org/scalapack/tools/descinit.f

PDGEMM: part 4

```
do i = 1, mb
  do j = 1, nb
    write(*,*) 'a(',i,',',j+2*my_id,')', a(i,j)
  enddo
enddo

do i = 1, mb
  do j = 1, nb
    write(*,*) 'b(',i,',',j+2*my_id,')', b(i,j)
  enddo
enddo

do i = 1, mb
  do j = 1, nb
    write(*,*) 'c(',i,',',j+2*my_id,')', c(i,j)
  enddo
enddo

call MPI_FINALIZE(rc)

stop
end
```

codes/scalapack/pmm/pmm.f90.4

pdgemm (TRANSA, TRANSB, M, N, K, ALPHA, A, IA, JA, DESCA, B, IB, JB, DESCB, BETA, C, IC, JC, DESCC)

- Result: `codes/scalapack/pmm/pmm_res.pdf`
- PDGEMM: `codes/scalapack/pmm/pdgemm.pdf`

```
2 call pdgemm ( 'N', 'N',    m,      n,      m, &
              1.0D0,   a,    1,      1, desca, &
4                 b,    1,    1, descb, 0.0D0, &
                  c,    1,    1, descc)
```

codes/scalapack/pmm/pmm.f90.3a