Efficient Languages, Codes and Parallelizing

Pascal M. Vaudrevange

07.Aug.2009

Efficient languages

- If all you have is a hammer, all problems look like nails
- Parsing through text files
- Extracting columns from files
- There is no right way, everything subjective (e.g. postscript plotter in F90)

My personal preferences

Problem	Language
vectors, matrices	Fortran or C++ (**NOT C**)
extracting data from text files	awk, sed, grep
math on extracted data from text files	perl, python
manipulation of many files/ filenames	bash

Examples: Bash

```
pascal@octonion:~/bash_test> ls
Cosmo.jpg  Schnitzel.jpg
pascal@octonion:~/bash_test> for i in `ls *.jpg`;do convert $i ${i/\.jpg/.png};done
pascal@octonion:~/bash_test> ls
Cosmo.jpg  Cosmo.png  Schnitzel.jpg  Schnitzel.png
pascal@octonion:~/bash_test>
```

- Advanced Bash Scripting Guide
  `http://tldp.org/LDP/abs/html/`
- convert is part of ImageMagick
  ps, png, pdf, jpg, avi, ...
- use regular expressions

Examples: awk

```
pascal@octonion:~/projects/circles/hpcc> cat scramble_open_45_prelim_bin.dat
0.313091 0.232143 14.9074 585
0.315435 0.5 32.1082 1260
0.317779 0.714286 45.8689 1800
0.320123 1 64.2164 2520
0.322468 0.714286 45.8689 1800
0.324812 0.803571 51.6025 2025
0.327156 0.732143 47.0156 1845
0.3295 0.767857 49.3091 1935
0.331844 0.321429 20.641 810
0.334189 0.321429 20.641 810
0.336533 0.142857 9.17378 360
0.338877 0.125 8.02706 315
0.341221 0.0892857 5.73361 225
0.343566 0.0178571 1.14672 45
0.34591 0.0535714 3.44017 135
0.348254 0.0714286 4.58689 180
0.350598 0 0 0
0.352942 0.0178571 1.14672 45
0.355287 0.0178571 1.14672 45
0.357631 0 0 0
pascal@octonion:~/projects/circles/hpcc> awk 'BEGIN {sum = 0} {sum=sum+$4} END{print sum}' scramble_open_45_prel
16740
pascal@octonion:~/projects/circles/hpcc> awk '{print $4}' scramble_open_45_prelim_bin.dat
585
1260
1800
2520
1800
2025
1845
1935
810
810
360
315
225
45
135
180
0
```

Examples: Perl

```
pascal@octonion:~/projects/cosmomc_may08> cat runMPI.pl
#!/usr/local/bin/perl
use Cwd;

#Use current directory as root
$cosmomc = cwd;

$params =  $ARGV[0];
$num = $ARGV[1];

$ini = $params;
if ($ini !~ m/\.ini/) {$ini= "$ini.ini"}

$path = $cosmomc;

open(Fout,">./scripts/script_MPI");
print Fout <<EMP;
#!/bin/csh -f
#PBS -N cosmomc
#PBS -l nodes=$num:ppn=2
#PBS -q workq
#PBS -r n
lamboot
cd $cosmomc
time mpirun N -O ./cosmomc $ini > ./scripts/$params.log
lamhalt
EMP
close(Fout);

chdir("./scripts");
@args=("qsub","./script_MPI");
system(@args);
chdir("../");

pascal@octonion:~/projects/cosmomc_may08> ./runMPI.pl params.ini 4
pascal@octonion:~/projects/cosmomc_may08> ./runMPI.pl params 4
```

Things to consider

- Does the language offer the features I need? C + Vectors = disaster
- What is your goal?
  - **NOT** to write a program that solves the problem
  - **BUT** to write a program that solves the problem and that **can be understood by you in 10 years**
- the shorter the program, the easier it is to understand

```
#!/usr/bin/perl
# 472-byte qrpff, Keith Winstein and Marc Horowitz <sipb-iap-dvd@mit.edu>
# MPEG 2 PS VOB file -> descrambled output on stdout.
# usage: perl -I <k1>:<k2>:<k3>:<k4>:<k5> qrpff
# where k1..k5 are the title key bytes in least to most-significant order

s''$/=\2048;while(<>){G=29;R=142;if((@a=unqT="C*",_)[20]&48){D=89;_=unqb24,qT,@
b=map{ord qB8,unqb8,qT,_^$a[--D]}@INC;s/...$/1$&/;Q=unqV,qb25,_;H=73;O=$b[4]<<9
|256|$b[3];Q=Q>>8^(P=(E=255)&(Q>>12^Q>>4^Q/8^Q))<<17,O=O>>8^(E&(F=(S=O>>14&7^O)
^S*8^S<<6))<<9,_=(map{U=_%16orE^=R^=110&(S=(unqT,"\xb\ntd\xbz\x14d")[_/16%8]);E
^=(72,@z=(64,72,G^=12*(U-2?0:S&17)),H^=_%64?12:0,@z)[_%8]}(16..271))[_]^((D>>=8
)+=P+(~F&E))for@a[128..$#a]}print+qT,@a}';s/[D-HO-U_]/\$$&/g;s/q/pack+/g;eval
```

But seriously

- **Goal**: short, readable, debuggable programs (ideally bug-free)
- Use libraries: no bugs that you can introduce
  - C++ Standard Template Library (STL)
    `http://www.cppreference.com/wiki/`
  - cfitsio
- Use what is fastest to deploy (after learning it)
- Think of time spent learning a new library/ language as investment in the future
- Use the language that is well-adjusted to the problem
- There is more (many more) than one way to do things

What problem do you want to solve?

Efficient Codes

I don’t know

What do you mean by efficient?

Efficient for humans

- **Write Short Functions**
- name the functions properly
- at most 7 things in short-term memory
- Don’t do everything in one language
- ”We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.” (D.Knuth)

Efficient to execute

- find the bottleneck (innermost loops)
- memory layout for arrays: C/C++ vs Fortran
- use a faster algorithm or library: e.g. libfftw vs. fftpack
- in Fortran: perform input/output in batch, not number-by-number
- don't copy large data structures, use pointers
- parallelize

Finding the bottleneck

add compiler switch `-g -pg`, recompile and run

```
pascal@octonion:~/projects/cranknich> gprof -bp ./cn.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 78.96     13.66    13.66 19702575     0.00     0.00  fdflux_
 10.23     15.43     1.77      202     0.01     0.06  __crankniching_MOD_leapforward
  6.76     16.60     1.17      304     0.00     0.00  __boundaryconditions_MOD_twodabsorbingbc
  2.66     17.06     0.46      101     0.00     0.05  __crankniching_MOD_evolveeuler
  0.87     17.21     0.15        6     0.03     0.03  __crankniching_MOD_energy
  0.23     17.25     0.04   311296     0.00     0.00  __boundaryconditions_MOD_get_boundary_values
  0.12     17.27     0.02   311296     0.00     0.00  __boundaryconditions_MOD_get_b
  0.06     17.28     0.01        1     0.01     0.01  __initialconditions_MOD_colliding_initialconditions
  0.06     17.29     0.01        1     0.01     0.02  __initialconditions_MOD_set_initialconditions
  0.06     17.30     0.01                             __crankniching_MOD_findphiminmax
  0.00     17.30     0.00   198147     0.00     0.00  __initialconditions_MOD_abic
  0.00     17.30     0.00      304     0.00     0.00  __boundaryconditions_MOD_set_boundaryconditions
  0.00     17.30     0.00       33     0.00     0.00  __inifile_MOD_tnamevaluelist_add
  0.00     17.30     0.00       23     0.00     0.00  __inifile_MOD_ini_namevalue_add
  0.00     17.30     0.00       23     0.00     0.00  __inifile_MOD_ini_read_string_file
  0.00     17.30     0.00       23     0.00     0.00  __inifile_MOD_tnamevaluelist_valueof
  0.00     17.30     0.00       13     0.00     0.00  __inifile_MOD_ini_read_real
  0.00     17.30     0.00       13     0.00     0.00  __inifile_MOD_ini_read_real_file
  0.00     17.30     0.00        9     0.00     0.00  __inifile_MOD_ini_read_int
  0.00     17.30     0.00        9     0.00     0.00  __inifile_MOD_ini_read_int_file
  0.00     17.30     0.00        2     0.00     0.00  __inifile_MOD_tnamevaluelist_setcapacity
  0.00     17.30     0.00        1     0.00    17.29  MAIN__
  0.00     17.30     0.00        1     0.00     0.00  __inifile_MOD_ini_close
  0.00     17.30     0.00        1     0.00     0.00  __inifile_MOD_ini_open
  0.00     17.30     0.00        1     0.00     0.00  __inifile_MOD_ini_open_file
  0.00     17.30     0.00        1     0.00     0.00  __inifile_MOD_ini_read_logical
  0.00     17.30     0.00        1     0.00     0.00  __inifile_MOD_ini_read_logical_file
  0.00     17.30     0.00        1     0.00     0.00  my_isnan_
pascal@octonion:~/projects/cranknich>
```

Parallelizing

- Shared Memory: OpenMP (easy)
- Distributed Memory: MPI (major rewrite)

OpenMP

```
pascal@octonion:~/projects/talks/programming_workshop/source_20090807> vi omp_test.f90
program omp_test
#ifdef _OPENMP
  use omp_lib
#endif
  implicit none
  integer(8) i, j, k, sum
  integer, parameter :: max=400
  sum=0
  !$omp parallel
  write(*,*) "Thread ", omp_get_thread_num(), " out of ", omp_get_num_threads(), " threads."
  !$omp do reduction(+:sum) private(i,j,k)
  do i=1,max; do j=1, max; do k=1,max
     sum=sum + i+j+k
  end do; end do; end do
  !$omp end do
  !$omp end parallel
  write(*,*) sum
end program omp_test

pascal@octonion:~/projects/talks/programming_workshop/source_20090807> /usr/bin/time ./omp_test
 Thread            2  out of            4  threads.
 Thread            0  out of            4  threads.
 Thread            1  out of            4  threads.
 Thread            3  out of            4  threads.
          38496000000
0.00user 0.00system 0:00.00elapsed 400%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+296minor)pagefaults 0swaps
pascal@octonion:~/projects/talks/programming_workshop/source_20090807> export OMP_NUM_THREADS=2
pascal@octonion:~/projects/talks/programming_workshop/source_20090807> /usr/bin/time ./omp_test
 Thread            1  out of            2  threads.
 Thread            0  out of            2  threads.
          38496000000
0.00user 0.00system 0:00.00elapsed 200%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+286minor)pagefaults 0swaps
pascal@octonion:~/projects/talks/programming_workshop/source_20090807>
```

OpenMP

```
pascal@octonion:~/projects/talks/programming_workshop/source_20090807> vi omp_test_c.c
#include<stdio.h>
#ifdef _OPENMP
#include<omp.h>
#endif
int main(int argc, char * argv[]){
  long i, j, k, sum;
  long max = 400;
  sum=0;
#pragma omp parallel
  {
#pragma omp critical
    {
      printf("Thread %i out of %i threads\n", omp_get_thread_num(), omp_get_num_threads());
    };
#pragma omp for reduction(+:sum) private(i,j,k)
    for(i=1; i<=max; i++)
      for(j=1; j<=max; j++)
for(k=1; k<=max; k++)
  sum+=i+j+k;
  };
  printf("%li\n", sum);
  return 0;
}
pascal@octonion:~/projects/talks/programming_workshop/source_20090807> make
gfortran -O2 -fopenmp -x f95-cpp-input -c omp_test_f.f90
gfortran -O2 -fopenmp omp_test_f.o -o omp_test_f
gcc -O2 -fopenmp -c omp_test_c.c
gcc -O2 -fopenmp omp_test_c.o -o omp_test_c
pascal@octonion:~/projects/talks/programming_workshop/source_20090807> /usr/bin/time ./omp_test_c
Thread 1 out of 4 threads
Thread 0 out of 4 threads
Thread 2 out of 4 threads
Thread 3 out of 4 threads
38496000000
0.08user 0.00system 0:00.02elapsed 296%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+221minor)pagefaults 0swaps
pascal@octonion:~/projects/talks/programming_workshop/source_20090807>
```

OpenMP

- thread: one (of possible several) instances/parts of a program that run concurrently and see the same address space/ the same variables
- careful not to write to the same variable from multiple threads “at the same time” – execution order is not guaranteed
- private vs. shared
- by default, all variables are visible for all threads
- reduction

private - shared - firstprivate

- *private* variables are NOT initialized, i.e. neither allocated nor set to a specific value $\Rightarrow$ initialized them before use!! (unless used with *copyin*)
- *private* variables are “local” variables to each thread and have different values in each thread
- *private* variables lose their values when exiting the parallel region (unless using *copyout*)
- *shared* variables are initialized and keep their value.
- *shared* variables point to the same memory address: only modify them if you can be sure that other threads don’t
- *firstprivate* variables are “local” to each thread, but are automaticcaly initialized with values from before the parallel region $\Rightarrow$ don’t use this for dynamically allocated scalars! (not supported for dynamical variables in Fortran)
- default setting is *shared*

reduction

- *reduction* is optional
- *reduction* understands $+, -, *, /$ (no overloaded operators yet?)

MPI - Message Passing Interface

- different processes (own memory, can be on different physical computers)
- pass variables through a network (slow)

Example

```
#ifdef MPI
  call MPI_init(ierr)
  call MPI_comm_rank(MPI_COMM_WORLD, MPIid, ierr)
  call MPI_comm_size(MPI_COMM_WORLD, MPIsize, ierr)
#endif

  if (iargc() .lt.1)then
     write(*,*) "Usage: circles <inifile>"
     stop
  end if

  call getarg(1,inifilename)
  call initialize(inifilename, nside_map)

  if (MPIsize .gt. 1) then
     outfilename=trim(adjustl(inifilename(1:len_trim(inifilename)-4)))//"_mpi_"//trim(adjustl(mpiid_string)) //"
  else
     outfilename=trim(adjustl(inifilename(1:len_trim(inifilename)-4)))//".dat"
  end if

  !loop over opening angle
  do iopening_angle=nside_search/2+mpiid, 2*nside_search, mpisize
.
.
.
  end do

  ! gather max S from all nodes and let node 0 do the writing to disk
#ifdef MPI
  call MPI_reduce(smax_angle, smax_result, 2*nside_search-nside_search/2, MPI_real, MPI_sum, 0, MPI_COMM_WORLD,
#endif
  if ((MPIsize .gt. 1) .and. (mpiid.eq.0)) then
     open(unit=full_outfile, file=trim(adjustl(inifilename(1:len_trim(inifilename)-4)))//".dat")
     do i=nside_search/2, 2*nside_search
        if  ((rad_to_deg(dble(theta_rings(i))) .lt. opening_angle_range%min) .or. (rad_to_deg(dble(theta_rings(i
        write(full_outfile,*) rad_to_deg(dble(theta_rings(i))), smax_result(i)
     end do
     close(full_outfile)
  end if

  call finalize()
```

MPI - functions

Command	Function
`MPI_Init`	Initialize
`MPI_comm_rank()`	ID of current program
`MPI_comm_size()`	number of processes
`MPI_reduce()`	analog of `reduce` clause in OpenMP
`MPI_send()`	send variable to a node
`MPI_recv()`	receive variable (needs to be matched up with `MPI_send()`)

MPI - Blocking

```
PROGRAM simple_send_and_receive
  INCLUDE 'mpif.h'
  INTEGER myrank, ierr, status(MPI_STATUS_SIZE)
  REAL a(100),b(100)
  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
  if( myrank.eq.0 )then
     do i=1,100
        a(i)=sqrt(real(i))
     end do
     call MPI_SEND( a, 100, MPI_REAL, 1, 17, MPI_COMM_WORLD, ierr)
  else if ( myrank.eq.1 )then
     call MPI_RECV( b, 100, MPI_REAL, 0, 17, MPI_COMM_WORLD, status, ierr )
  endif
  call MPI_FINALIZE(ierr)
END
```

```
#include <stdio.h>
#include <mpi.h>
#include <math.h>
int main (int argc, char **argv) {
  int myrank,i;
  MPI_Status status;
  double a[100],b[100];
  MPI_Init(&argc, &argv);  /* Initialize MPI */
  MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* Get rank */
  if( myrank == 0 )        /* Send a message */
  {
    for (i=0;i<100;++i)
       a[i]=sqrt(i);
    MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
  }
  else if( myrank == 1 )   /* Receive a message */
    MPI_Recv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );

  MPI_Finalize();          /* Terminate MPI */
}
```

MPI - Nonblocking

```
PROGRAM simple_deadlock_avoided
INCLUDE 'mpif.h'
INTEGER myrank, ierr, status(MPI_STATUS_SIZE)
INTEGER request
REAL a(100), b(100)
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
if( myrank.eq.0 )then
   call MPI_IRECV( b, 100, MPI_REAL, 1, 19, MPI_COMM_WORLD, request, ierr )
   call MPI_SEND( a, 100, MPI_REAL, 1, 17, MPI_COMM_WORLD, ierr)
   call MPI_WAIT( request, status, ierr )
else if ( myrank.eq.1 )then
   call MPI_IRECV( b, 100, MPI_REAL, 0, 17, MPI_COMM_WORLD, request, ierr )
   call MPI_SEND( a, 100, MPI_REAL, 0, 19, MPI_COMM_WORLD, ierr)
   call MPI_WAIT( request, status, ierr )

endif
call MPI_FINALIZE(ierr)
END
```

```
#include <stdio.h>
#include <mpi.h>
void main (int argc, char **argv) {
int myrank;
MPI_Request request;
MPI_Status status;
double a[100], b[100];
MPI_Init(&argc, &argv);  /* Initialize MPI */
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* Get rank */
if( myrank == 0 ) {
  MPI_Irecv( b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &request );
  MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
  MPI_Wait( &request, &status );
}
else if( myrank == 1 ) {
  MPI_Irecv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &request );
  MPI_Send( a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
  MPI_Wait( &request, &status );
}
MPI_Finalize();          /* Terminate MPI */
}
```

Online Tutorials

`http://ci-tutor.ncsa.illinois.edu/` for MPI