INTRODUCTION TO PARALLEL COMPUTING

MARTIN ČUMA AND COLTON GRAINGER (SCRIBE)

1. Options for parallelization

	single data	multiple data
single instruction	SISD	SIMD
multiple instruction		MIMD

1.1. **Shared memory.** Simpler programming, native with multi-core processors.

Shared memory is an efficient means of passing data between processes. In a shared-memory model, parallel processes share a global address space that they read and write to asynchronously. Asynchronous concurrent access can lead to *race conditions*, and mechanisms such as locks, semaphores and monitors can be used to avoid these. `https://en.wikipedia.org/wiki/Parallel_programming_model`

We'll focus on three elements of parallelization in this talk.

processes	threads	fibers
carries state information	shares state and memory resources, subsets of processes	"lightweight" threads that *yield*

Fibers explicitly "yield" to allow other fibers to run; "yielding" is a form of computer multitasking in which the operating system never initiates a context switch from a running process to another process. `https://en.wikipedia.org/wiki/Thread_(computing)`

Forms of shared memory parallelization.

- POSIX "pthreads", `https://en.wikipedia.org/wiki/POSIX_Threads`, low-level
  - "pthreads defines a set of C programming language types, functions and constants"
- OpenMP and OpenACC, *good for beginners*, mid-level

1.2. **Distributed memory.**

- MPI (Message Passing Interface), the *de facto standard* for HPC

  MPI is meant to provide essential virtual topology, synchronization, and communication functionality between a set of processes (that have been mapped to nodes/servers/computer instances) in a language-independent way, with language-specific syntax (bindings), plus a few language-specific features.

2. OpenMP, profiling, libraries

Consider vector addition $\vec{z} = a\vec{x} + \vec{y}$ in FORTRAN.

Date: 2019-05-21.
url: `http://www.chpc.utah.edu/~mcuma/rmacc/`.

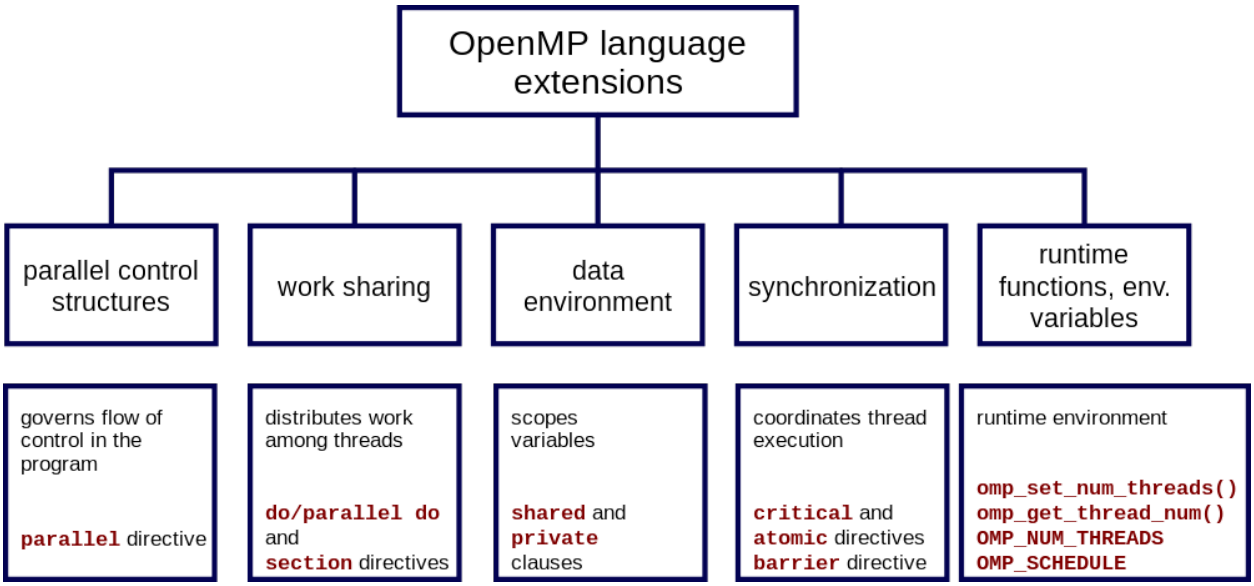

FIGURE 1. Chart of OpenMP constructs

```
subroutine saxpy_serial(z, a, x, y, n)
integer i, n
real z(n), a, x(n), y(n)

do i=1, n
    z(i) = a*x(i) + y(i)
enddo
return
```

Here's the same process with OpenMP parallelization into threads.

```
subroutine saxpy_parallel_omp(z, a, x, y, n)
integer i, n
real z(n), a, x(n), y(n)

!$omp parallel do
do i=1, n
    z(i) = a*x(i) + y(i)
enddo
return
```

Then executed in the console:

```
$ gcc -fopenmp saxpy.f
$ export OMP_NUM_THREADS=16
$ ./a.out
```

2.1. **Tips for OpenMP.**

> Dependencies among statements or instructions may hinder parallelism — parallel execution of multiple instructions, either by a parallelizing compiler or by a processor exploiting instruction-level parallelism. Recklessly executing multiple instructions without considering related dependences may cause danger of getting wrong results, namely hazards. `https://en.wikipedia.org/wiki/Data_dependency`

For example, suppose we had a loop with iterations given by:

```
x = a(i)
b(i) = c + x
a(i) = a(i+1) + x
x += a(i)
```

OpenMP can't fix this.

Hence, try to write programs and *manage your data dependencies*.

- Create private (thread only) variables, e.g., to handle Runge-Kutta methods, or matrix multiplication.
- Rearrange dependencies.
- Reduce over loops, e.g., sum over threads.
- Setup Makeflow to visualize parallel processes. `https://github.com/cooperative-computing-lab/makeflow-examples`

3. OpenACC

Here's an implementation of the above process in OpenACC.

```
subroutine saxpy_parallel_oacc(z, a, x, y, n)
integer i, n
real z(n), a, x(n), y(n)

!$acc kernels datain(x,y) dataout(z)
do i=1, n
    z(i) = a*x(i) + y(i)
enddo
return
```

Here's the console execution.

```
$ pgcc -acc -Minfo=accel saxpy.f
$ pgaccelinfo #to verify that GPU is available
$ ./a.out
```

3.1. Warnings for OpenACC.

- Need to data dependencies (Like in OpenMP)
- Data locality
  - Transfers from host to GPU and back take time
  - try to minimize transfers
  - `#pragma acc data [copyin, copyout, create,...]`
- Parallel regions
  - More explicit execution control (warps, threads)
  - `#pragma acc parallel`
- Procedure calls
  - If procedure is executed on the GPU
  - `#pragma acc routine`

4. MPI

Here's the process implemented with MPI functions.

```
subroutine saxpy_parallel_mpi(z, a, x, y, n)
integer i, n, ierr, my_rank, nodes, i_st, i_end
real z(n), a, x(n), y(n)
```

```
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,my_rank,ierr)
call MPI_Comm_size(MPI_COMM_WORLD,nodes,ierr)
i_st = n/nodes*my_rank+1
i_end = n/nodes*(my_rank+1)

do i=i_st, i_end
    z(i) = a*x(i) + y(i)
enddo
call MPI_Finalize(ierr)
return
```

4.1. **Tips for MPI.**

- Try AVX `https://en.wikipedia.org/wiki/Advanced_Vector_Extensions` for intel hardware 10–0 years old?
- Try the Intel MPI Library, via (a free) academic license.
- Set yourself up!
  - define an MPI header (for either fortran or C)
  - learn basic MPI functions `Init, Finalize, Size, Rank`
    - e.g., `MPI_Comm_size` determines the size of the group associated with a communicator
    - `MPI_Comm_rank` determines the rank of the calling process in the communicator
- decompose tasks
  - e.g., frequencies in wave solvers
- decompose domains
  - e.g., distribute atoms in molecular dynamics
  - e.g., distribute mesh in ODE/PDE solvers
- other features
  - one sided communication
  - point-to-point communication
  - collective communication
  - blocking and non-blocking communication
  - derived data types
  - process topologies
  - parallel I/O
  - dynamic processes

5. Libraries

- BLAS, LAPACK, linear algebra routines
- MKL, ACML, hardware vendor libraries
- PETSc
  - `https://www.mcs.anl.gov/petsc/index.html`
- Trilinos
  - `https://trilinos.github.io/about.html`

  The Trilinos Project is an effort to develop and implement robust algorithms and enabling technologies using modern object-oriented software design, while still leveraging the value of established libraries such as PETSc, Metis/ParMetis, SuperLU, Aztec, the BLAS and LAPACK. It emphasizes abstract interfaces for maximum flexibility of component interchanging, and provides a full-featured set of concrete classes that implement all abstract interfaces.

  A core requirement of many engineering and scientific applications is the need to solve linear and non-linear systems of equations, eigensystems and other related problems. Thus it is no

surprise that any part of the application that solves these problems is called a "solver." The exact definition of what specifically constitutes a solver depends on many factors. However, a good working definition of a solver is the following: *any piece of software that finds unknown values for some set of discrete governing equations in an application.* Another characteristic of solvers is that we can often implement them in such a way that they are "general-purpose", so that the details of how the discrete problem was formed are not specifically needed for the solver to work (although information about problem characteristics can often be vital to robust solutions.) Advanced solution techniques increasingly require more than "just a matrix" since the matrix alone discards so much information about the origin of the problem and how it might best be solved. As a result solvers need compatible tools in differentiation, discretization and data partitioning. These tools are also essential for robust solutions in scalable computing environments.

- For python, do install NumPy from Anaconda. (Intel MKL support?)
- For R, try Future?

6. Summary

Even in the case of shared memory, parallel programming is very difficult: getting quickly correct results is very challenging. The problem revolves around communications between parts of the program: it's easy to lock everything (to get correct results) but then efficiency suffers a lot. Fortunately, many problems can be handled with some simplified approaches: the fork-join principle that underlines OpenMP and the data parallelism that appears in foreach. The latter principle is particularly adapted to many learning tasks, for example. `http://apiacoa.org/teaching/big-data/smp.en.html`

7. More resources

- XSEDE web courses, `https://www.xsede.org/web/xup/course-calendar`
- Petascale Computing Institute videos, `https://www.youtube.com/XSEDETraining`