MOVING PARALLEL WITH OPENMP

Juan Pablo Mallarino

mandaro.m@gmail.com

July 15, 2021

Agradecimientos a la Universidad del Rosario

the big question

Computation

What is NOT OpenMP?

the big question

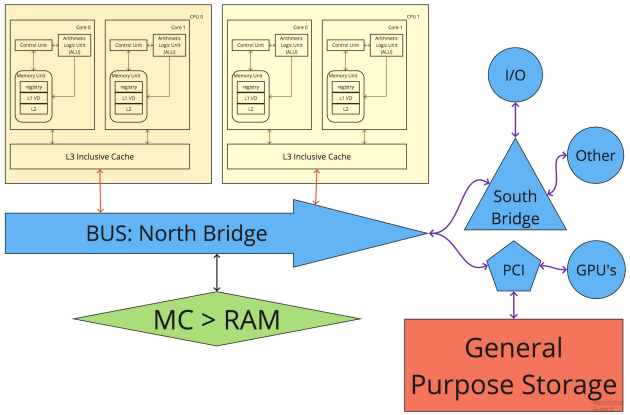


Figure: Von Neumann Architechture

Computation

What is NOT OpenMP?

Key infrastructure components

- Storage
- RAM
- Processing block: registries, instruction sets and clock
- FPGA's, GPU's, accelerators and other alternate processing units (RaspBerries, portable devices . . . ARM)
- Compilers - Translator to Machine language

the big question

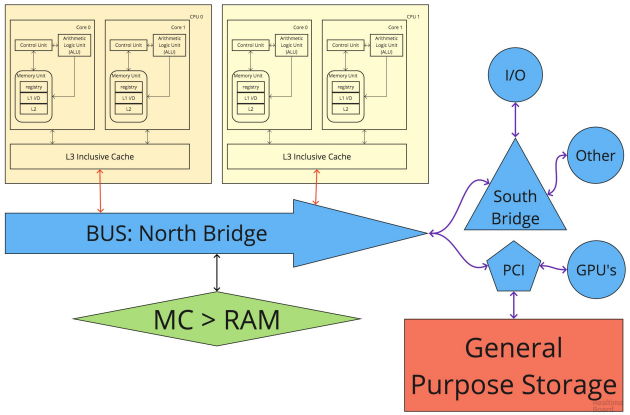


Figure: Von Neumann Architechture

Computation

What is NOT OpenMP?

Key infrastructure components

- Storage
- RAM
- Processing block: registries, instruction sets and clock
- FPGA's, GPU's, accelerators and other alternate processing units (RaspBerries, portable devices . . . ARM)
- Compilers - Translator to Machine language

Limitations & Complications

1. All of the above
2. Education: infrastucture topology, coding strategies, profiling & optimization
3. Interpreted languages
4. Unix like systems
5. Time - accelerating technologies and real-time applications
6. Threats $\Longrightarrow$ Cybersecurity (`https://meltdownattack.com/`)

the big question

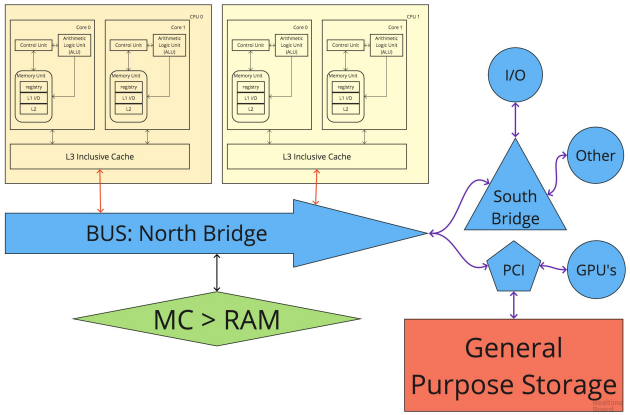


Figure: Von Neumann Architechture

History of parallelism

1. Origin dates back to the 80's
2. ILP + Vectorization: the superscalar architecture
3. Memory complexity:

CPU	$\sim 0.5ns$	$1\times$

Table: By Jeff Dean@Google: http://research.google.com/people/jeff/

4. Memory coherency: *(i.)* HW with ECC *(ii.)* Software
5. Memory topology: UMA & NUMA & *cc*-NUMA

the big question

History of parallelism

7. 1996 SGI bought CRAY and soon after formed the ARB. 1997 OpenMP was born and announced at the New York Times.
8. CPU processor development stalled: *(i.)* Quantum limit $\sim 9nm$ *(ii.)* Energy efficiency per FLOP kept dropping
9. A full scale development of *multi*-core processors
10. Later: GPGPU's & MIC's & FPGA's

Basics

Definition

A parallel computer is a system that is able to execute simultaneously multiple processing elements cooperatively to solve a computational problem

Requirements

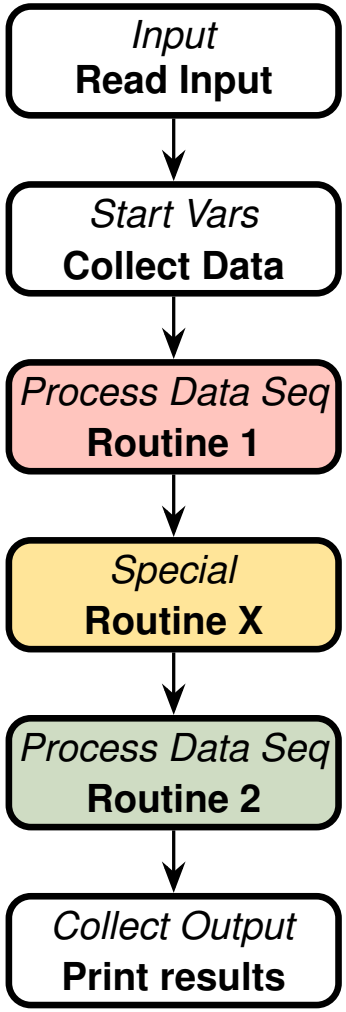
- Hardware
- OS
- Libraries: PThreads, TBB, OpenMP
- For HPC: Understand the process and data distribution model

Terminology

- △ Concurrent: A program is one in which multiple tasks can be *in progress* at any instant. Or in *multiple*-THREADS!
- △ Parallel: A program is one in which multiple tasks *cooperate closely* to solve a problem.
- △ Distributed: A program may need to cooperate with other programs to solve a problem.

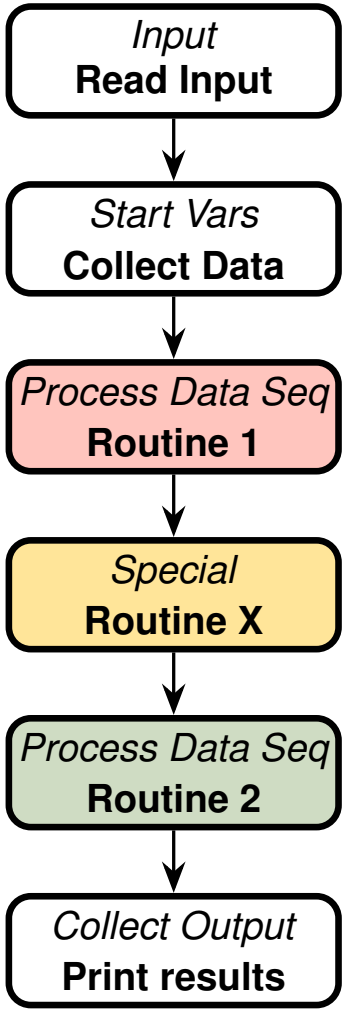
Pacheto **An instroduction to Parallel Programming**, Elsevier (2011)

The idea?



- The idea is to identify opportunities of parallelism
- Develop the application to exploit parallelism
- Run the application: identify Bugs and Improvements
- Use resources efficiently

The idea?

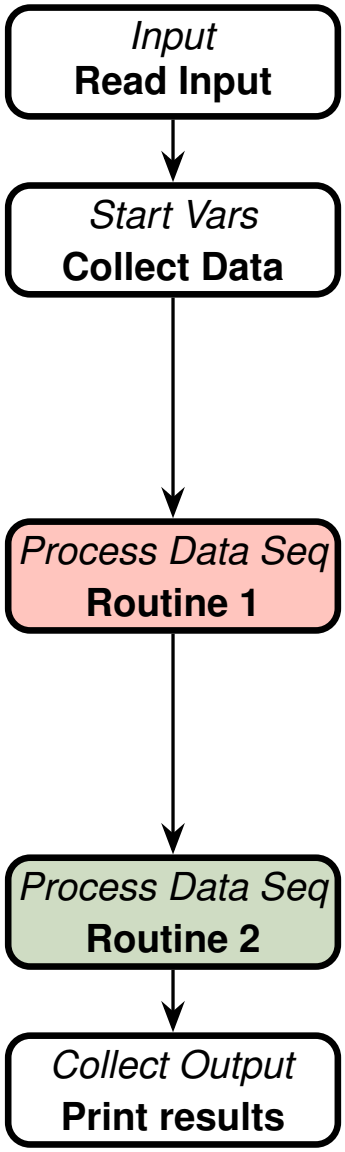


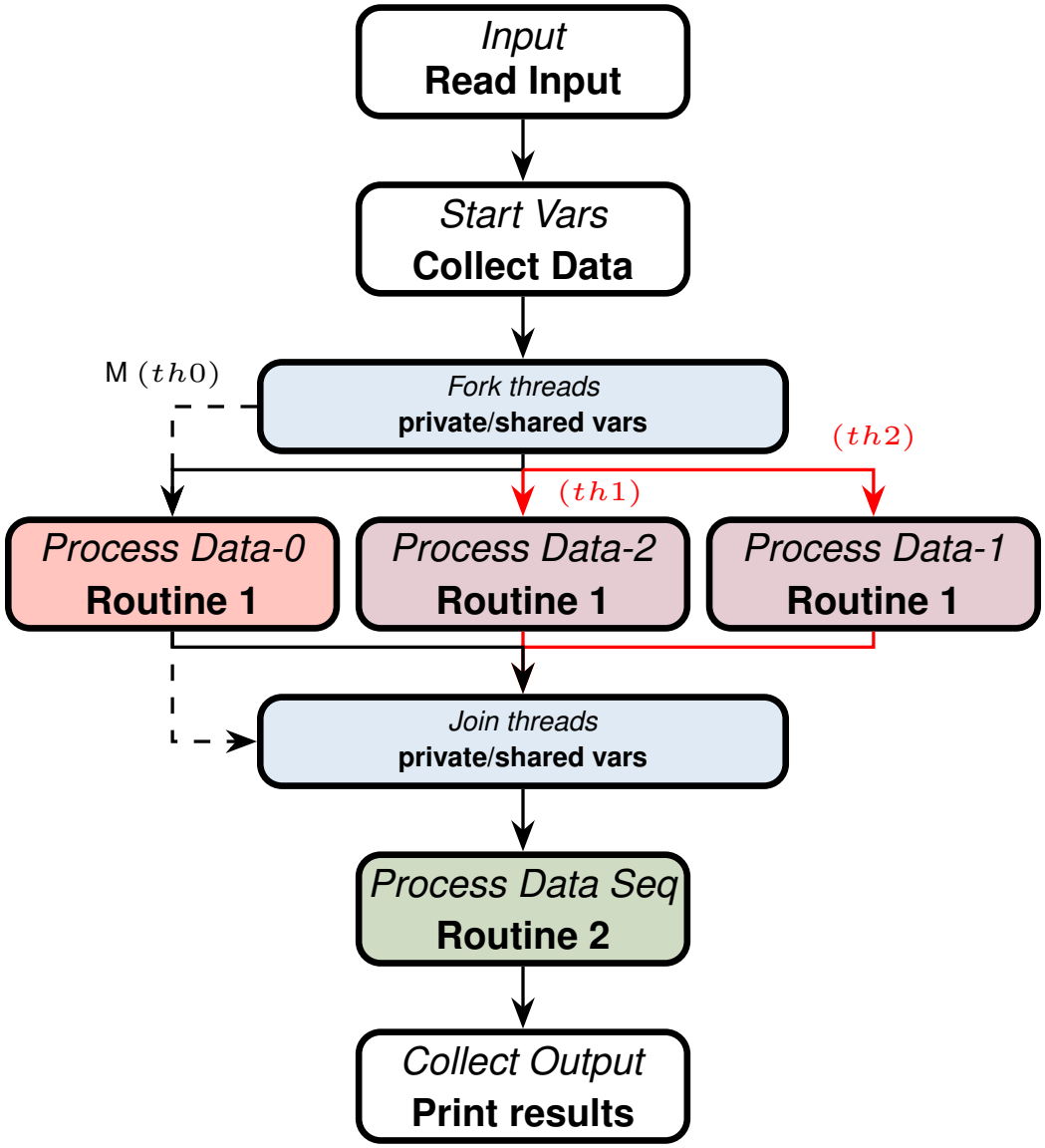
- The idea is to identify opportunities of parallelism
- Develop the application to exploit parallelism
- Run the application: identify Bugs and Improvements
- Use resources efficiently

API and Libraries

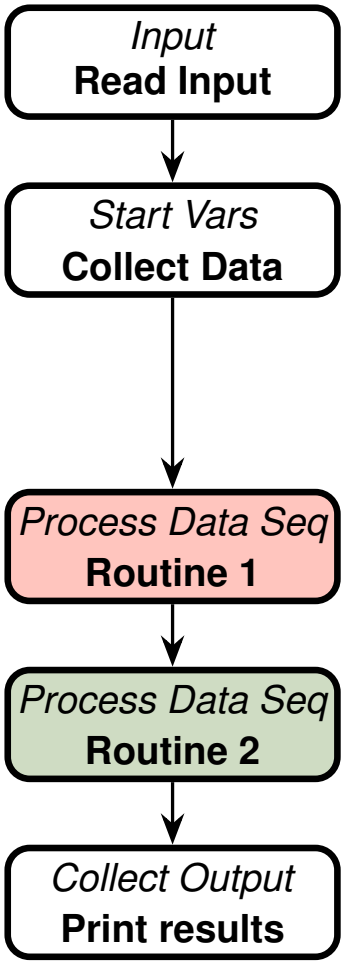
- ⋆ Pthreads (HARD): routines & variables
- ⋆ OpenMP: pragmas, routines & variables
- ⋆ Comes for C/C++ and Fortran

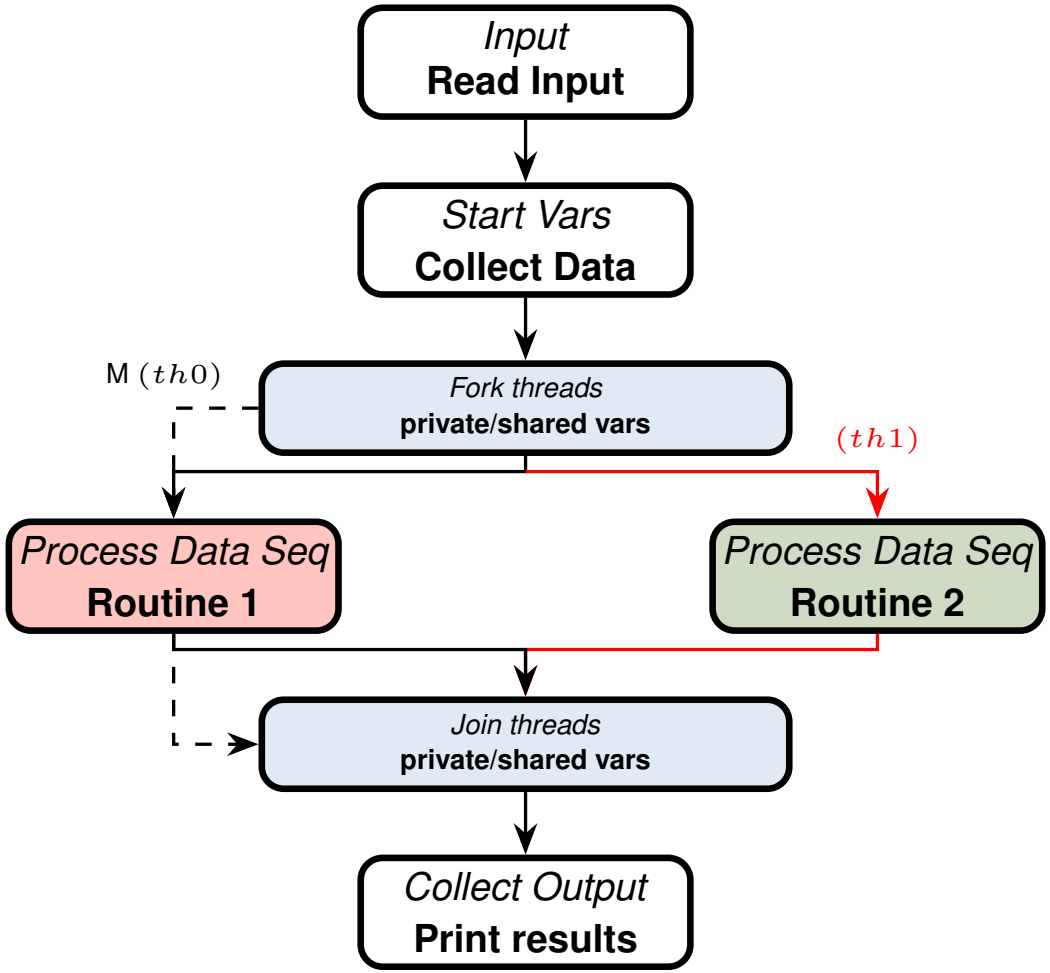
The idea parallelized: Data





The idea parallelized: Task





Ahmdal's Law

Due to the overheads, parallelism is only achieved at a certain level. Let t be the execution time of a sequential application, then

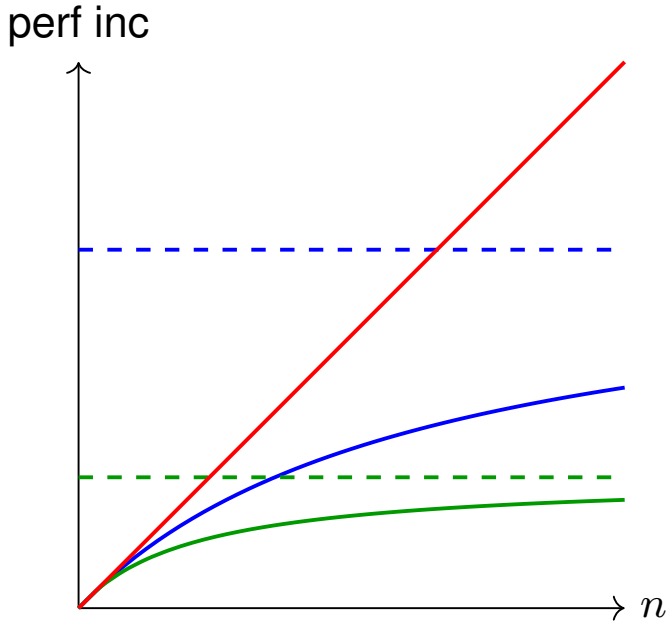
$$t = t_s + t_p,$$

with t_s and t_p the time of sequential and parallel parts. If the data or tasks are going to be split equally into n threads, then the new time t' is written as,

$$t' = t_s + n\,\delta t_{\text{oh}} + \frac{t_p}{n},$$

where δt_{oh} is the overhead time required to fire up each thread (approximately linear). Defining $f := \frac{t_s}{t_p}$,

$$\text{perf inc} = \frac{t}{t'} = n\frac{1+f}{1+n\,f+n^2\delta t_{\text{oh}}} \leqslant 1 + \frac{1}{f}$$



Open Multi Processing: preliminaries

Data environment/Declarations

1. `#pragma omp`
2. `#pragma omp threadprivate`
3. `shared/*private`
4. clauses
5. `master`/`threads`

Parallel construct structures

1. `omp parallel`

clauses / options

1. $\mathrm{if}(\dots)$
2. $\mathrm{num\_threads}(\dots)$
3. $\mathrm{private}(\dots)$
4. $\mathrm{shared}(\dots)$
5. $\mathrm{firstprivate}(\dots)$:
   $x(t=0) = x$ before construct
6. $\mathrm{default}(\dots)$: `none|shared`
7. $\mathrm{copyin}(\dots)$:
   x private on master copied to threads privates
8. $\mathrm{reduction}(operator|list)$

How do we compile?

OpenMP version

"As of GCC 4.2, the compiler implements version 2.5 of the OpenMP specification, as of 4.4 it implements version 3.0 and since GCC 4.7 it supports the OpenMP 3.1 specification. GCC 4.9 supports OpenMP 4.0 for C/C++, GCC 4.9.1 also for Fortran. GCC 5 adds support for Offloading."

GCC: `gcc -std=c++11 ex6-loop-reduction.cpp -o exe -fopenmp`

Intel Compiler: `icc -std=c++11 ex6-loop-reduction.cpp -o exe -qopenmp`

Simple example #1: ex1-hostname

Source Code 1: Printing hostname with Master or Single!

```
1  #define INFO_BUFFER_SIZE 1024
2  int main(void){
3      InfoOpenMP();
4
5      char hostname[INFO_BUFFER_SIZE];
6      char username[INFO_BUFFER_SIZE];
7      #pragma omp parallel
8      {
9      gethostname(hostname, INFO_BUFFER_SIZE);
10     getlogin_r(username, INFO_BUFFER_SIZE);
11     printf("Hostname %s in thread
         ↪   %d\n",hostname,omp_get_thread_num());
12     printf("Username %s in thread
         ↪   %d\n",username,omp_get_thread_num());
13     } /*-- End of parallel region --*/
14     return 0;
15 }
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Delared the parallel construct
5. We are using `printf`
6. Prints the thread numbers $[0, 1, 2, 3]$

Simple example #1: ex1-hostname

Source Code 2: Printing hostname with Master or Single!

```
1  #pragma omp parallel
2  {
3  gethostname(hostname, INFO_BUFFER_SIZE);
4  getlogin_r(username, INFO_BUFFER_SIZE);
5  cout<<"Hostname "<<hostname<<" in thread
     ↪   "<<omp_get_thread_num()<<endl;
6  cout<<"Username "<<username<<" in thread
     ↪   "<<omp_get_thread_num()<<endl;
7  } /*-- End of parallel region --*/
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Delared the parallel construct
5. We are using `printf`
6. Prints the thread numbers $[0, 1, 2, 3]$

Problem

1. Disordered output! `std::cout`
2. How does `cout` work? What is `flush`?
3. Can you attempt to write a single `cout` and `flush`?

Reference Guide!

Look at the reference guide!

Open Multi Processing: preliminaries II

Data environment/Declarations

1. `#pragma omp`
2. `#pragma omp threadprivate`
3. `shared/*private`
4. clauses
5. `master` /`threads`

Work sharing

1. `omp [parallel] for`
2. `omp [parallel] sections`
3. Only Fortran: `omp [parallel] workshare`
4. `omp single`

loop clauses / options

Can be combined with the *parallel* construct

1. $\mathrm{private}(\dots)$
2. $\mathrm{firstprivate}(\dots)$
3. $\mathrm{lastprivate}(\dots)$:
   $x(t = t_f) = x$ last "loop" value
4. $\mathrm{reduction}(operator|list)$
5. $\mathrm{ordered}(\dots)$:
   if ordered construct inside parallel region!
6. $\mathrm{schedule}(kind[, chunk\_size])$:
   static,dynamic,guided,runtime,auto
7. nowait:

sections clauses / options

Can be combined with the *parallel* construct
private+firstprivate)+lastprivate)+
reduction+nowait

single clauses / options

1. private+firstprivate)+nowait
2. $\mathrm{copyprivate}(\dots)$: x private in thread to threads

Open Multi Processing: preliminaries II

Data environment/Declarations

1. `#pragma omp`
2. `#pragma omp threadprivate`
3. `shared/*private`
4. clauses
5. `master` /`threads`

atomic clauses / options

1. read(...)
2. write(...)
3. update(...)
4. capture(...)

Synchronization

1. `omp barrier`:
   Look semaphores
2. `omp ordered`:
   only for *loops*!
3. `omp critical` $[(name)]$:
   is a block!
   Could use hints!
4. `omp atomic`: only a statement < 3.1!
5. `omp master`:
   no barrier at the end!
6. `omp flush`: enforces data consistency
   relaxed consistency model
7. `omp task`:
   called from within *single* construct
8. `omp taskwait`

Open Multi Processing: preliminaries II

Data environment/Declarations

1. `#pragma omp`
2. `#pragma omp threadprivate`
3. `shared/*private`
4. clauses
5. `master` /`threads`

Data environment

1. *Routines* (or functions)
2. ENVIRONMENT VARIABLES

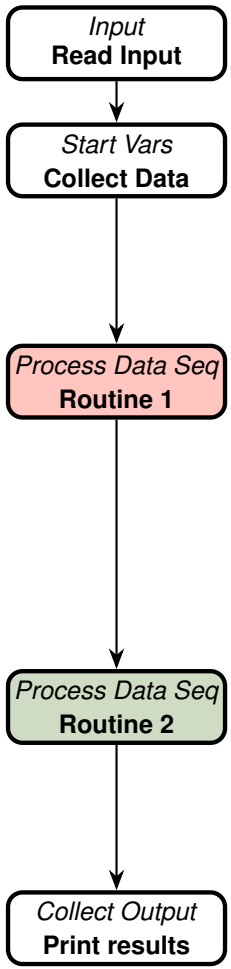
useful routines

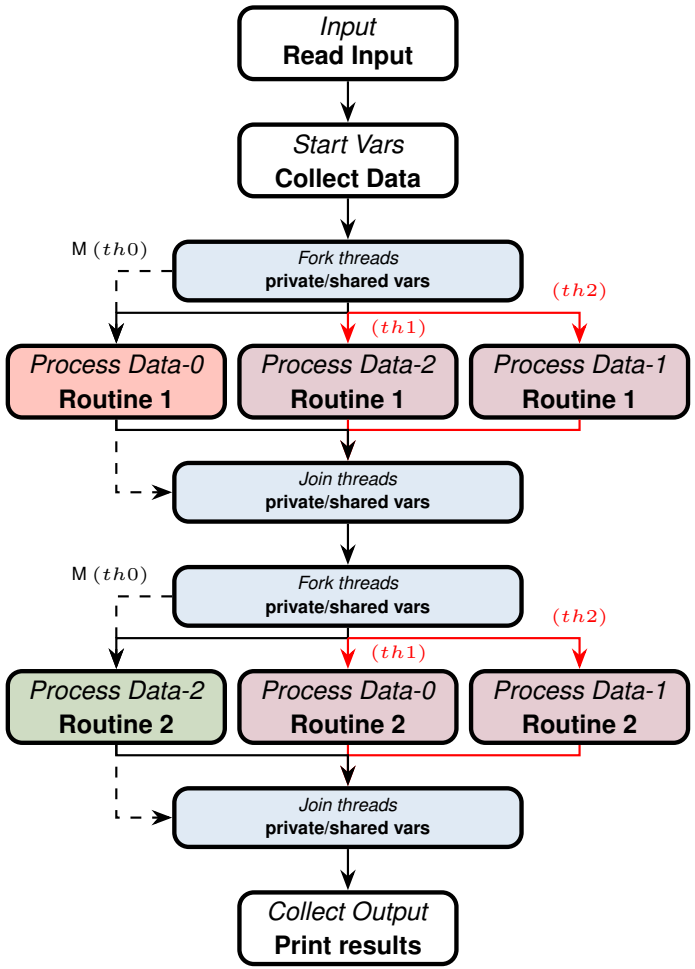
1. `omp_set_max_threads()`
2. `omp_get_max_threads()`
3. `omp_get_num_threads()`
4. `omp_get_num_devices()`
5. `omp_get_thread_num()`
6. `omp_get_thread_limit()`
7. `omp_set_nested()`
8. `omp_get_nested()`
9. `omp_in_parallel()`
10. *LOCKS!*

useful *ENV VARS*

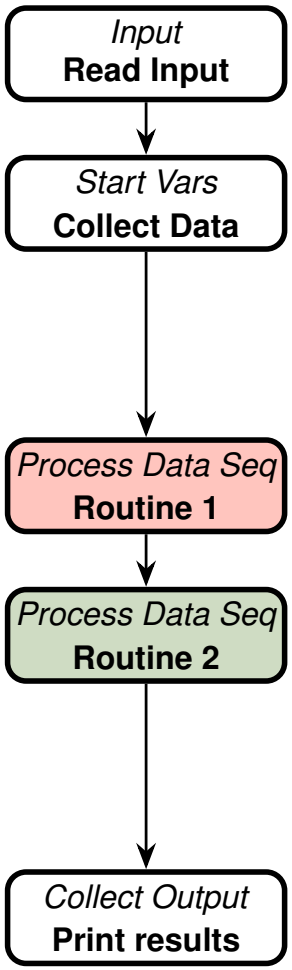
1. `OMP_THREAD_LIMIT`
2. `OMP_NUM_THREADS`
3. `OMP_DYNAMIC`
4. `OMP_NESTED`

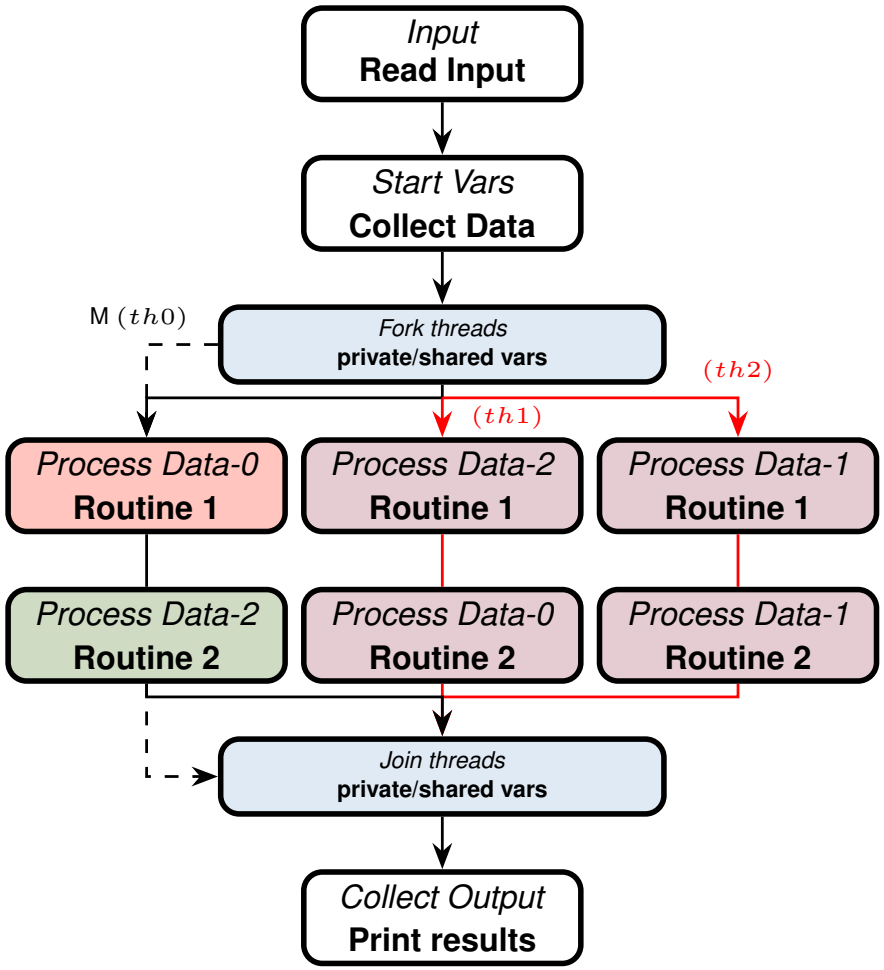
Interesting scenarios!



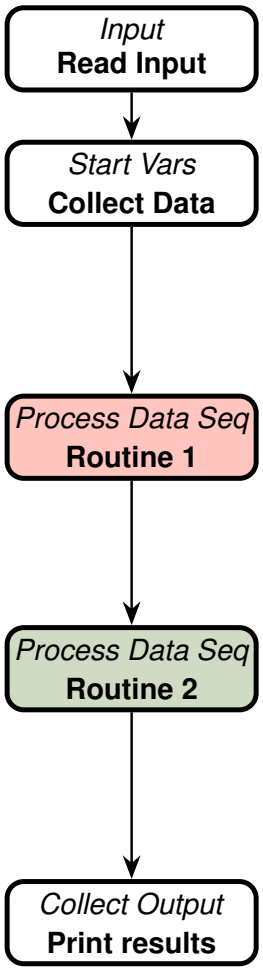


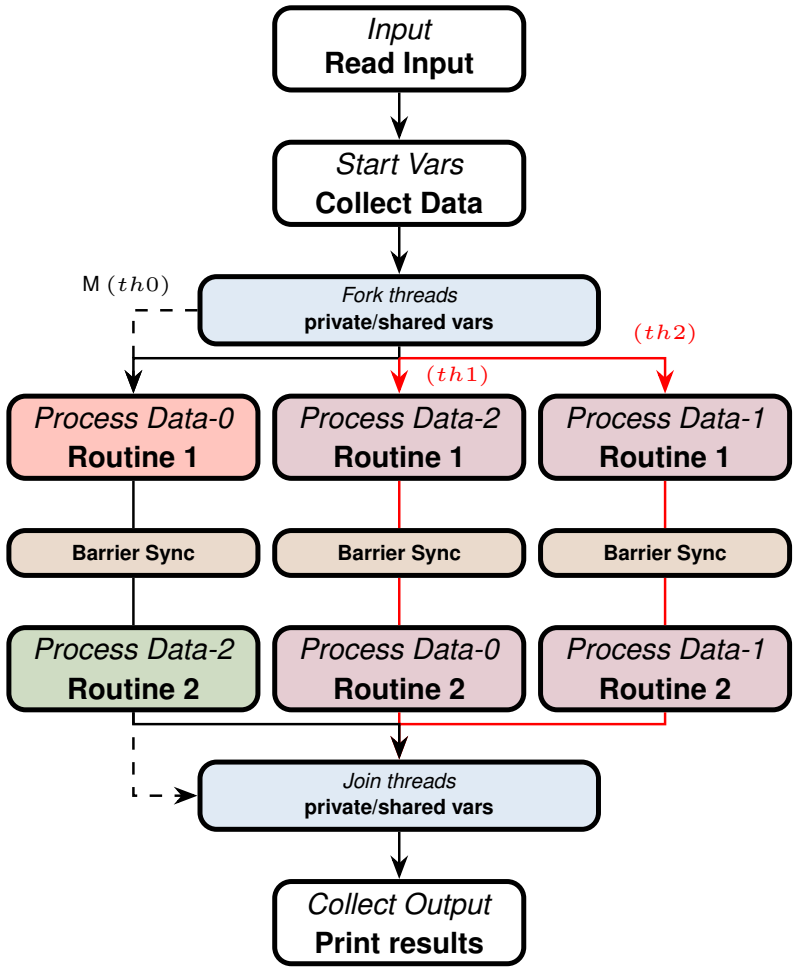
Interesting scenarios!



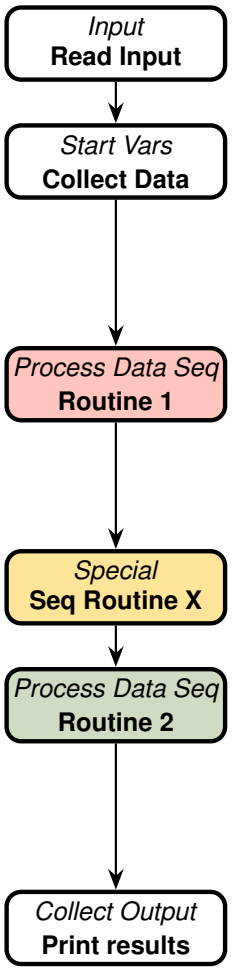


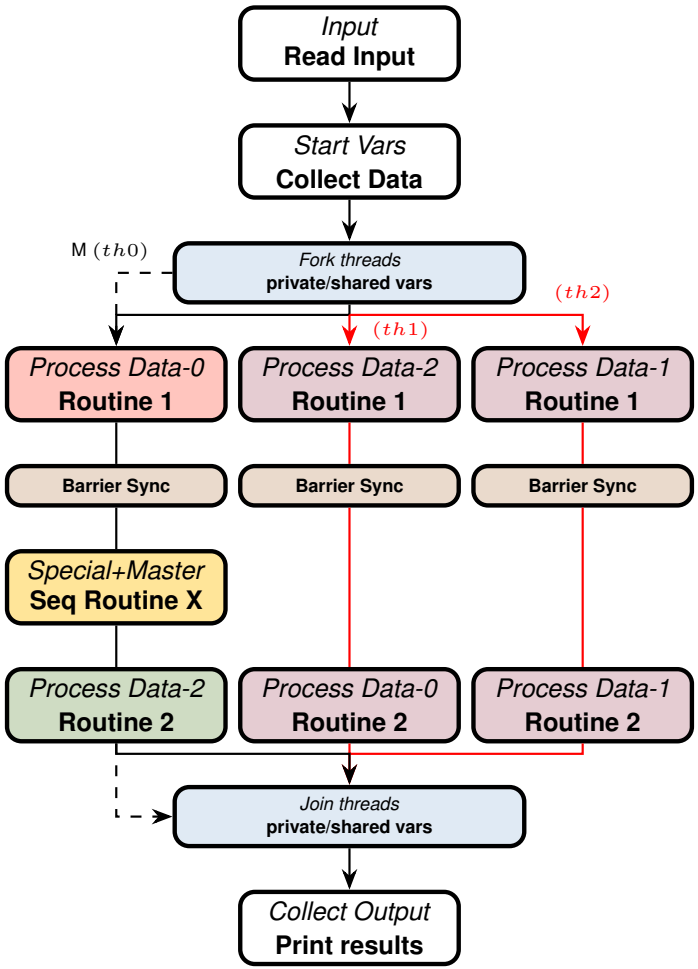
Interesting scenarios!



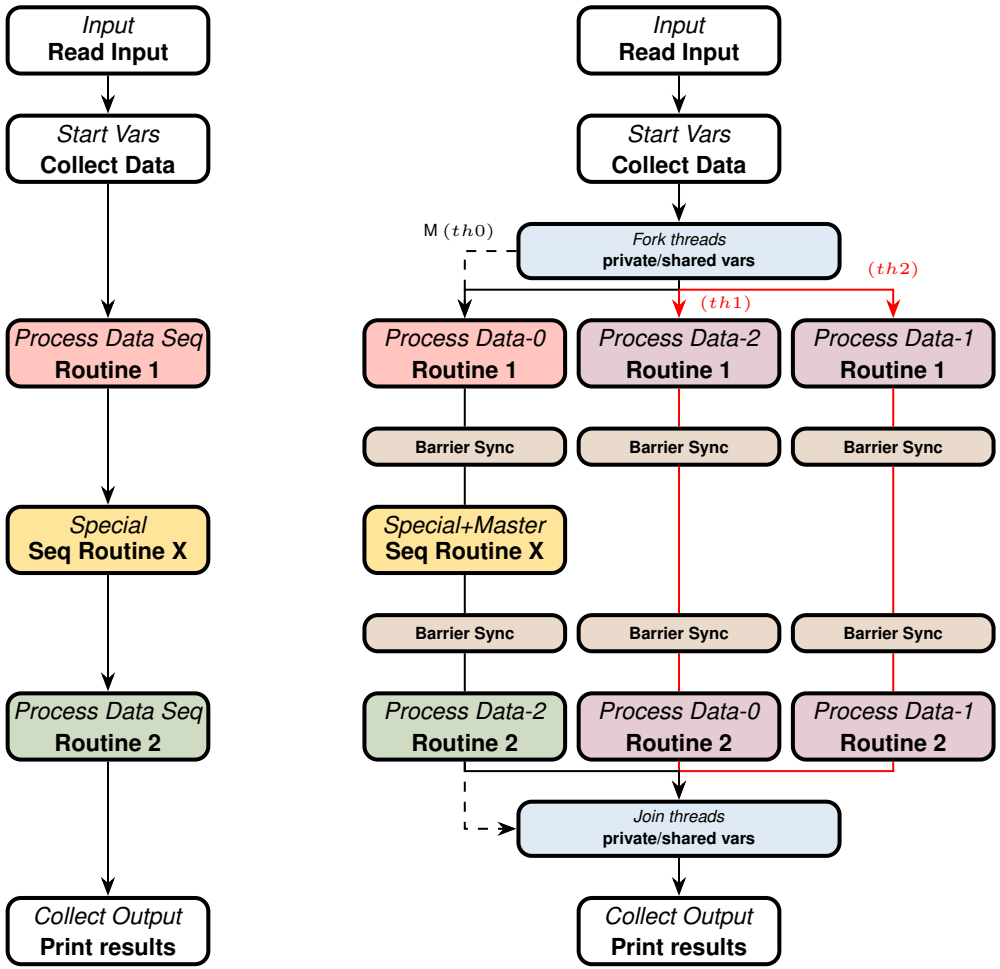


Interesting scenarios!

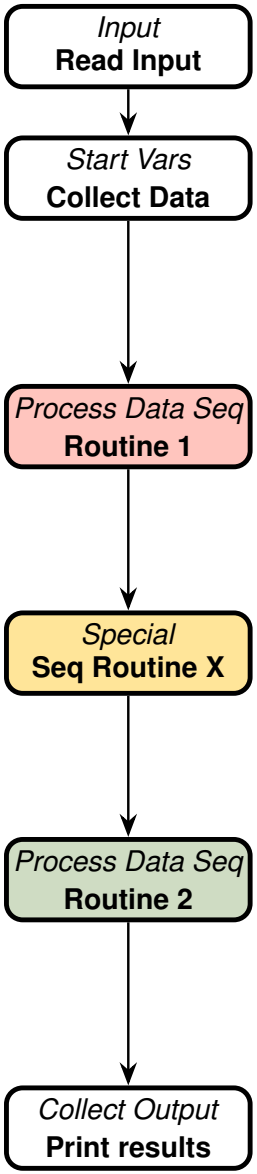


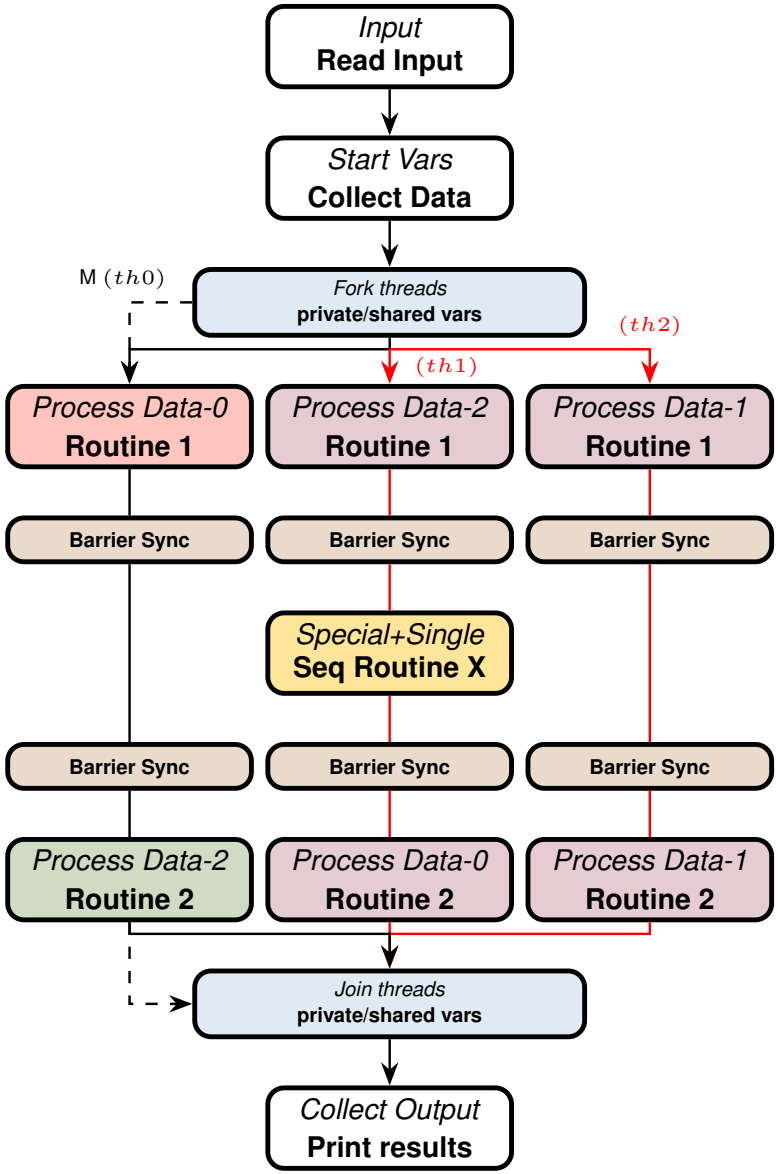


Interesting scenarios!



Interesting scenarios!





Down to business !

Simple example #2: ex2-master-single

Source Code 3: Printing thread number and Sleep!

```
1  int main(void){
2  #ifdef _OPENMP
3      printf("**MESSAGE** OpenMP enabled\n");
4      (void) omp_set_dynamic(FALSE);
5      (void) omp_set_num_threads(4);
6  #else
7      printf("**MESSAGE** OpenMP disabled\n");
8  #endif
9      #pragma omp parallel
10     {
11         int TID = omp_get_thread_num();
12         sleep(omp_get_thread_num());
13
14         printf("In parallel region - Thread ID is
               ↪  %d\n",TID);
15     } /*-- End of parallel region --*/
16     return 0;
17 }
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Sets the sleep according to thread #

Simple example #2: ex2-master-single

Source Code 4: Printing thread number and Sleep!

```
1   int extra_time = 0;
2   #pragma omp parallel shared(extra_time)
3   {
4       int TID = omp_get_thread_num();
5       #pragma omp master
6       {
7           printf("\tInside Block - Thread ID is
              ↪  %d\n",TID);
8           sleep(1);
9           extra_time = 1;
10      }
11      sleep(TID+extra_time);
12      printf("In parallel region - Thread ID is
          ↪  %d\n",TID);
13  } /*-- End of parallel region --*/
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Sets the sleep according to thread #

Problem

1. Cause: value not updated soon enough

Simple example #2: ex2-master-single

Source Code 5: Printing thread number and Sleep!

```
1   int extra_time = 0;
2   #pragma omp parallel shared(extra_time)
3   {
4       int TID = omp_get_thread_num();
5       #pragma omp single
6       {
7           printf("\tInside Block - Thread ID is
              ↪  %d\n",TID);
8           sleep(1);
9           extra_time = 1;
10      }
11      sleep(TID+extra_time);
12      printf("In parallel region - Thread ID is
          ↪  %d\n",TID);
13  } /*-- End of parallel region --*/
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Sets the sleep according to thread #

Problem

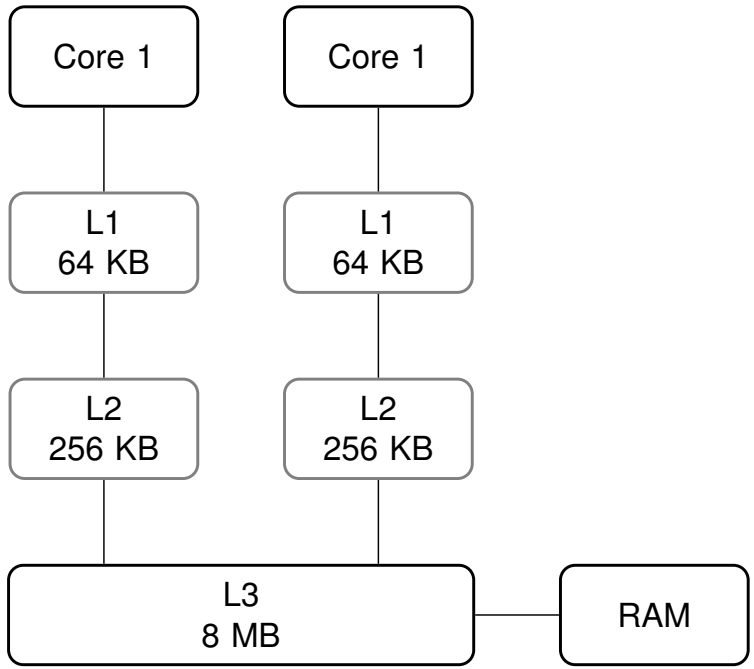
1. Corrected

I

nspect `ex2-vector.cpp`

What is the issue?

- Each thread is a copy of the original $th0$
- Private variables have redundant "private" memmory addresses
- Used values are load to cache! Not all cache is shared
- Updates have to be enforced for data coherency!
- How? `flush` or `barrier`.



Simple example #3: ex3-copy

Source Code 6: Private vars and firstprivate!

```
1   int main(void){
2       InfoOpenMP();
3
4   int x = 10;
5   printf("thread %d original value:
      ↪  %d\n",omp_get_thread_num(),x);
6   #pragma omp parallel private(x)
7   {
8       int TID = omp_get_thread_num();
9       printf("thread %d value: %d\n",TID,x);
10  } /*-- End of parallel region --*/
11  printf("thread %d after value:
      ↪  %d\n",omp_get_thread_num(),x);
12  return 0;
13  }
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Prints zero inside parallel construct!
5. Prints again 10 after parallel construct!

Problem

1. Private value restarts at ZERO!

Simple example #3: ex3-copy

Source Code 7: Private vars and firstprivate!

```
1    #pragma omp parallel firstprivate(x)
2    {
3        int TID = omp_get_thread_num();
4        printf("thread %d value: %d\n",TID,x);
5    } /*-- End of parallel region --*/
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Prints zero inside parallel construct!
5. Prints again 10 after parallel construct!

Problem

1. Corrected, prints $x = 10$.

Special Exercise

Use `copyprivate` to copy private variable from `master` or `single` to other threads.
Can you use `threadprivate`?

Simple example #4: ex4-shared

Source Code 8: Updating variable!

```
1  int main(void){
2      InfoOpenMP();
3
4      int x = 10;
5      #pragma omp parallel
6      {
7          int TID = omp_get_thread_num();
8          x += 5;
9          printf("thread %d value: %d\n",TID,x);
10     } /*-- End of parallel region --*/
11     printf("Value: %d\n",x);
12     return 0;
13 }
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. All threads add to x
5. x is shared

Problem

1. Every execution prints different values
2. RACE CONDITION
3. Think about all previous exercises!

Simple example #5: ex5-loop

Source Code 9: Adding arrays!

```
1  int main(void){
2      InfoOpenMP();
3      // ...
4      // Create two engines
5      default_random_engine gen1(sseq);
6      mt19937_64 gen2(sseq);
7      // Now define and parametrize two distributions
8      uniform_real_distribution<float> unif(-1, 1);
9      normal_distribution<float> normal(0, 1);
10
11     SimpleTimer _t;
12     float v1[N]; // Note array is in the stack!
13     _t.start("Random fill");
14     for(int it; it < N; it++)
15         v1[it] = normal(gen2); // Random noise
16     _t.stop(); _t.print();
17         return 0;
18 }
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. No parallel construct
5. Prints the arrays $v1$, $v2$ and $v1 + v2$

Simple example #5: ex5-loop

Source Code 10: Adding arrays!

```
1    _t.start("Adding up vectors!");
2    #pragma omp parallel shared(j,N,v1,v2,res)
3    {
4        #pragma omp for
5        for(int j = 0; j < N; j++)
6            res[j] = v1[j] + v2[j];
7    } /*-- End of parallel region --*/
8    _t.stop(); _t.print();
```

Problems

1. Any comments?

Additional exercises

1. Create another vector v2 with uniform data
2. Create function that sums two vectors into res (openMP)
3. Create function that computes the mean and variance (openMP)
4. Create function that computes inner product (openMP)
5. Create function that computes the norm product (openMP)
6. Check that the variance has to be 4/3 for res (openMP)
7. PRO: Fill vectors v1 and v2 using parallel threads (random vector)

Simple example #5: ex5 norm-al problems

Source Code 11: Norm of a vector!

```
1   Decimal norm_v1;
2   #pragma omp parallel shared(j,N,norm_v1,v1)
3   {
4       #pragma omp for
5       for(j = 0; j < N; j++)
6           norm_v1 += v1[j]*v1[j];
7   } /*-- End of parallel region --*/
8   // OR...
9   #pragma omp parallel for shared(j,N,norm_v1,v1)
10  for(j = 0; j < N; j++)
11      norm_v1 += v1[j]*v1[j];
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Parallel workshare construct
5. Prints the quadratic norm of array $|v1| \simeq \sqrt{N}$

Problem

1. Norm doesn't provide reliable results!

Simple example #5: ex5 with critical

Source Code 12: Norm of a vector!

```
1   Decimal norm_v1;
2   #pragma omp parallel shared(N,norm_v1,v1)
3   {
4       Decimal tmp = 0;
5       #pragma omp for
6       for(j = 0; j < N; j++){
7           tmp = v1[j]*v1[j];
8           #pragma omp critical
9           {
10              norm_v1 += tmp;
11          }
12      }
13  } /*-- End of parallel region --*/
14  // OR...
15  #pragma omp parallel for shared(N,norm_v1,v1)
16  for(j = 0; j < N; j++)
17      #pragma omp critical
18      norm_v1 += v1[j]*v1[j];
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Parallel workshare construct
5. Prints the quadratic norm of array $|v1| \simeq \sqrt{N}$

Problem

1. Solved with: critical

Simple example #5: ex5 with atomic

Source Code 13: Norm of a vector!

```
1   Decimal norm_v1;
2   #pragma omp parallel shared(N,norm_v1,v1)
3   {
4       Decimal tmp = 0;
5       #pragma omp for
6       for(j = 0; j < N; j++){
7           tmp = v1[j]*v1[j];
8           #pragma omp atomic
9           norm_v1 += tmp;
10      }
11  } /*-- End of parallel region --*/
12  // OR...
13  #pragma omp parallel for shared(N,norm_v1,v1)
14  for(j = 0; j < N; j++)
15      #pragma omp atomic
16      norm_v1 += v1[j]*v1[j];
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Parallel workshare construct
5. Prints the quadratic norm of array $|v1| \simeq \sqrt{N}$

Problem

1. Solved with: atomic

Simple example #5: ex5 with ordered

Source Code 14: Norm of a vector!

```
1   #pragma omp parallel shared(N,norm_v1,v1)
2   {
3       double tmp = 0;
4       #pragma omp for ordered
5       for(j = 0; j < N; j++){
6           tmp = v1[j]*v1[j];
7           #pragma omp ordered
8           {
9               norm_v1 += tmp;
10          }
11      }
12  } /*-- End of parallel region --*/
13  // OR...
14  #pragma omp parallel for shared(N,norm_v1,v1)
      ↪  ordered
15  for(j = 0; j < N; j++)
16      #pragma omp ordered
17      norm_v1 += v1[j]*v1[j];
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Parallel workshare construct
5. Prints the quadratic norm of array $|v1| \simeq \sqrt{N}$

Problem

1. Solved with: ordered

Simple example #5: ex5 with reduction

Source Code 15: Norm of a vector!

```
1   #pragma omp parallel shared(N,norm_v1,v1)
2   {
3       #pragma omp for reduction(+:norm_v1)
4       for(j = 0; j < N; j++)
5           norm_v1 += v1[j]*v1[j];
6   } /*-- End of parallel region --*/
7   // OR...
8   #pragma omp parallel for shared(N,norm_v1,v1)
      ↪  reduction(+:norm_v1)
9   for(j = 0; j < N; j++)
10      norm_v1 += v1[j]*v1[j];
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. Parallel workshare construct
5. Prints the quadratic norm of array $|v1| \simeq \sqrt{N}$

Problem

1. Solved with: reduction

Simple example #6: ex6-loop-matrix-reduction

Source Code 16: Norm of a vector!

```
1  int main(void){
2      InfoOpenMP();
3      // ...
4
5      SimpleTimer _t;
6      float A1[N][M]; // Note array is in the stack!
7      VectorMemUsage(N*M*sizeof(float),"A1");
8      for(int j = 0; j < N; j++)
9          for(int k = 0; k < M; k++)
10             A1[j][k] = normal(gen2); // Random noise
11
12     _t.start("Matrix Multiply");
13     #pragma omp parallel for
14     for(int j = 0; j < N; j++)
15         for(int k = 0; k < N; k++)
16             for(int l = 0; l < M; l++){
17                 res[j][k] += A1[j][l]*A2[l][k];
18             }
19     _t.stop(); _t.print();
20         return 0;
21 }
```

Notice

1. `InfoOpenMP`
2. Check the `defs.h`!
3. Manually setting the # of threads?
4. No parallel construct
5. Prints the quadratic norm of array $|v1| \simeq \sqrt{N}$

Exercise 1

Problem statement: Normalize array $v1/\sqrt{|v1|}$ into $v1$

Exercise 2

Problem statement: Find the determinant and inverse of a matrix

Exercise 3

Problem statement: Integrate an arbitrary real 1D function

Exercise 4

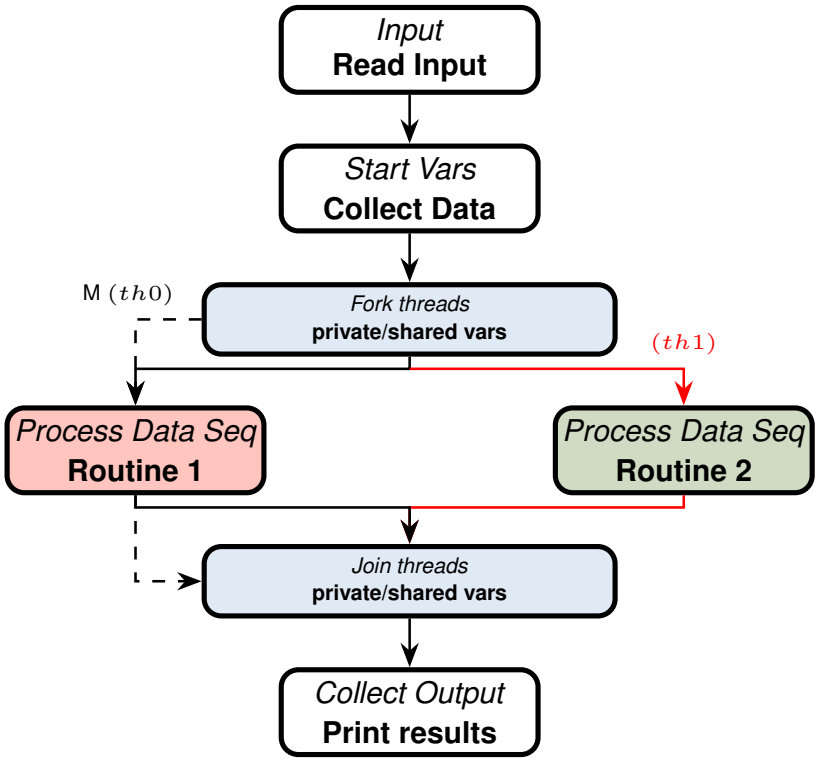
Problem statement: Evaluate the Fourier Transform

$$c_k = \frac{1}{2\pi} \int f(x) e^{-i\ k\ x} \mathrm{d}x \tag{1}$$

Exercise 5

Sort an array of floats or integers! Create a `swap` function that takes two parameters (pass by reference) and does the swapping. Then write the Sequential code. Finally, parallelize it! What is the order of the problem? *Hint:* N^p.

Back to the parallelized Task idea: ex7-task-palindrome/ex8-fibonacci



Source Code 17: The word palindrome!

```
1   printf("A ");
2   printf("race ");
3   printf("car ");
4   printf("is fun to watch.\n");
5
6   #pragma omp parallel
7   {
8       #pragma omp single
9       {
10          printf("A ");
11          #pragma omp task
12          { printf("race "); }
13          #pragma omp task
14          { printf("car "); }
15          #pragma omp taskwait
16          printf("is fun to
              ↪  watch.\n");
17      }
18  }
```

Exercise 6

Problem statement: Find random numbers using all threads

1. `nesting`: delicate!
2. `cancel`: The analog of break or exception handling
3. `simd`
4. `device`: For Coprocessors, GPU's and FPGA devices, or ...

Bibliography

Special thanks to ALL of you!

- **USING OpenMP - The Next Step**, *Ruud van der Pas, et. al.*, MIT Press (2017)
- **Introduction to Parallel Programming**, *Peter S. Pacheco*, Elsevier (2011)
- **Deep Learning**, *Ian Goodfellow, et. al.*, MIT Press (2016)
- **Structured Parallel Programming**, *Michael McCool, et. al.*, Elsevier (2012)