High-Performance Scientific Computing

Introduction to Parallel Programming

Ned Nedialkov

McMaster University

10 January 2023

Outline

Processes

- Process: instance of a program
- Consists of
  - executable
  - stack
  - heap
  - file descriptors
  - information about which resources can be accesses
  - information about the state: ready, running, waiting, blocked
    see e.g. ▸ Process state
    Try the `top` command
- Multitasking
  - many processes execute on a CPU (core)
  - the CPU switches between them

Threads

- smallest sequence of instructions that can be managed independently by a scheduler
- typically a thread is part of a process
- multiple threads can live within the same process
- share resources of a process, e.g. memory
- on a single CPU, it switches between threads
- on multiple CPUs/cores, threads can execute in parallel

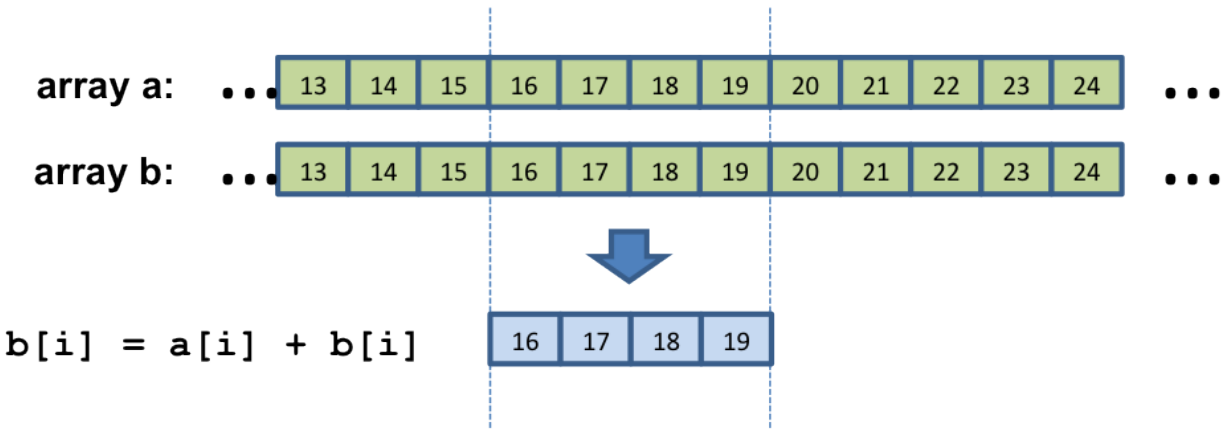
Flynn's taxonomy

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
- Multiple instruction streams, single data stream (MISD), uncommon
- Multiple instruction streams, multiple data streams (MIMD)

SIMD

Single Instruction Multiple Data

- The same instruction is applied to multiple data items.
- Intel, AMD processors
  - Streaming SIMD Extensions (SSE), 2 operands
  - Advanced Vector Extensions (AVX), 4 or 8 operands
- GPUs



Consider

```
// addvec.c
void add_vec(int n, double* a, double* b, double* c) {
#pragma vector always // pragma for the Intel compiler to vectorize
  for (int i = 0; i < n; i++)
    c[i] = a[i] + b[i];
}
```

With Intel's C compiler,

```
icx -c -O3 -qopt-report=2 -qopt-report-file=stderr addvec.c
```

produces

```
Global optimization report for : add_vec

LOOP BEGIN at addvec.c (4, 3)
<Multiversioned v2>
    remark #15319: Loop was not vectorized: novector directive used
LOOP END

LOOP BEGIN at addvec.c (4, 3)
<Multiversioned v1>
    remark #25228: Loop multiversioned for Data Dependence
    remark #15300: LOOP WAS VECTORIZED
    remark #15305: vectorization support: vector length 2
LOOP END

LOOP BEGIN at addvec.c (4, 3)
<Remainder loop for vectorization>
LOOP END
}
```

Adding the `-mavx` option

```
icx -c -O3 -mavx -qopt-report=2 -qopt-report-file=stderr addvec.c
```

```
LOOP BEGIN at addvec.c (4, 3)
<Multiversioned v1>
    remark #25228: Loop multiversioned for Data Dependence
    remark #15300: LOOP WAS VECTORIZED
    remark #15305: vectorization support: vector length 4
LOOP END
```

Try `gcc -fopt-info-vec-all -O3 addvec.c`

See also ▸ Using the Vectorizer

SPMD

Single Program Multiple Data

- the same program executes as different processes
- a process knows its ID, usually called rank
- suppose a program P has

```
if (my_rank == 0) {
   A()
 }
else {
   B()
 }
```

If an instance of P executes as process 0, A() executes otherwise B() executes

Example: simple MPI program

```
/* Send a message from all processes with rank != 0 to process 0.
   Process 0 prints the messages received.
 */
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char* argv[]) {
  int     my_rank;   /* rank of process */
  int     p;         /* number of processes */
  int     source;    /* rank of sender */
  int     dest;      /* rank of receiver */
  int     tag = 0;   /* tag for messages */
  char    message[100]; /* storage for message */
   keywordstyleMPI_Status status; /* status for receive */
```

```
/* Initialize MPI */
 keywordstyleMPI_Init(&argc, &argv);
/* Find out process rank */
 keywordstyleMPI_Comm_rank( keywordstyleMPI_COMM_WORLD, &my_rank);
/* Find out number of processes */
 keywordstyleMPI_Comm_size( keywordstyleMPI_COMM_WORLD, &p);
if (my_rank != 0) {
  printf("PE %d sends to PE 0: Hello from process %d \n", my_rank,
       my_rank);
  /* Create message */
  sprintf(message, "Hello from process %d", my_rank);
  dest = 0;
  /* Use strlen+1 so that '\0' gets transmitted */
   keywordstyleMPI_Send(message, strlen(message) + 1,
           keywordstyleMPI_CHAR, dest, tag,  keywordstyleMPI_COMM_WORLD);
```

```
  }
  else { /* my_rank == 0 */
    for (source = 1; source < p; source++) {
       keywordstyleMPI_Recv(message, 100,
              keywordstyleMPI_CHAR, source, tag,
              keywordstyleMPI_COMM_WORLD, &status);
      printf("PE 0 receives from PE %d: %s\n", source, message);
    }
  }
  /* Shut down MPI */
   keywordstyleMPI_Finalize();
  return 0;
}
```

- Compile

```
mpicc mpi-greetings.c
```

- Execute on 4 processes

```
mpirun -np 4 ./a.out
```

Example: simple OpenMP program

```
#include <omp.h>
#include <stdio.h>
int main()
{
  int nthreads, tid;
/* Start threads */
#pragma omp parallel private(nthreads, tid)
  {
    /* Obtain thread number */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);
    /* Only master thread does this */
    if (tid == 0)
      {
        nthreads = omp_get_num_threads();
        printf("Number of threads = %d\n", nthreads);
      }
  } /* All threads join master thread */
}
```

- Compile

```
gcc -fopenmp openmp-hello.c
```

  Run

```
./a.out
```

- In bash

```
export OMP_NUM_THREADS=8
./a.out
```

- In tcsh, csh

```
setenv OMP_NUM_THREADS 8
```