Primers

- Microprocessor design

  faster single processor (high power consumption, circuit complexity)

  vs. <u>multicore processor</u>

- Serial (sequential) program → $\boxed{\text{auto\_converter}}$ → parallel program

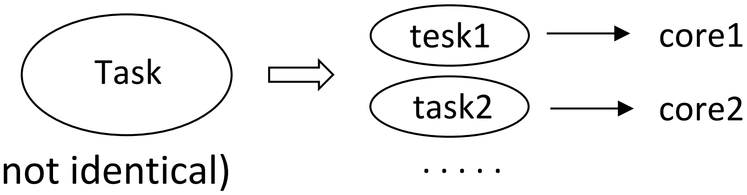
  not successful (very inefficient)

  vs. designing a new parallel algo/implementation

  much better way of achieving HP.

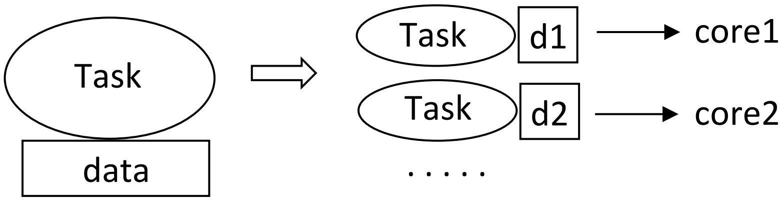
- Parallel program

  <u>task parallel</u>



  divided tasks (not identical)

  vs. <u>data parallel</u>



  identical tasks with a portion of data each

  Issues: processes need coordination

  Communication – share info/data, send/receive msg

  Synchronization – wait for proper order

  Load balancing – reduce the critical path

<u>Terms</u> ($\nexists$ clear boundary)

Concurrent – multiple tasks progress simultaneously

Parallel – multiple tasks cooperate closely to solve a problem
Time/HP are critical

Distributed – a program may need to cooperate with other programs to solve a problem, e.g., client/server model

- Parallel programming paradigms

  using explicit parallel constructs (ex, MPI, OpenMP) – more efficient

  vs. higher level parallel languages – less efficient

  ex) C++ extensions with

  - PThread (Posix Thread) – on shared memory systems
    libraries of type definition, functions, macros

  - OpenMP – shared memory system (e.g., multicore, SMP)
    library and some modifications to C/C++ compiler

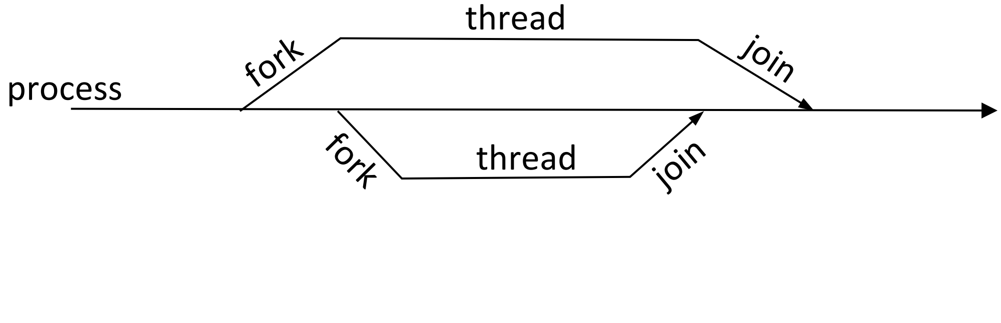
  - MPI – msg passing on distributed memory systems (e.g., clusters)
    libraries of type definition, functions, macros

- Process/thread (light weight)

  Execution switches betwwen threads faster than between processes;

  Threads belonging to the same process share resources (e.g., mem, I/O);

  Each thread has its own PC and stack;


thread
process
fork
join
fork
thread
join