Parallel Programming with OpenMP

Edmund Sumbar
WestGrid Support Staff

- OpenMP...

  a way to exploit **multi-processor** and **multi-core** hardware in an attempt to speedup program execution

- OpenMP…

  a way to exploit **multi-processor** and **multi-core** hardware in an attempt to speedup program execution

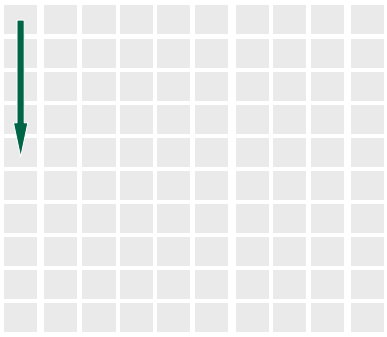
  …a sort of **code optimization**

- Consider this common type of optimization

```
do i = 1, N
    do j = 1, M
        a(i,j)...
    end do
end do
```

- Software accesses array elements by row

- Hardware stores array elements by column (Fortran)



- Reorder loop nest (if possible) for more efficient memory access

```
do i = 1, N
    do j = 1, M
        a(i,j)...
    end do
end do
```

original (slow)

```
do j = 1, M
    do i = 1, N
        a(i,j)...
    end do
end do
```

optimized (fast)

- Adapting your program to the hardware can yield a performance improvement

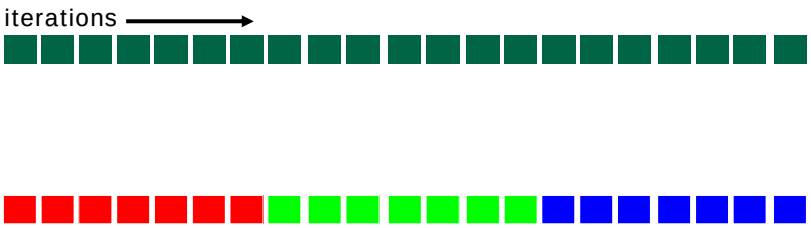
- Adapting your program to the hardware can yield a performance improvement
- Apply the same philosophy to multi-processor and multi-core hardware…consider a loop

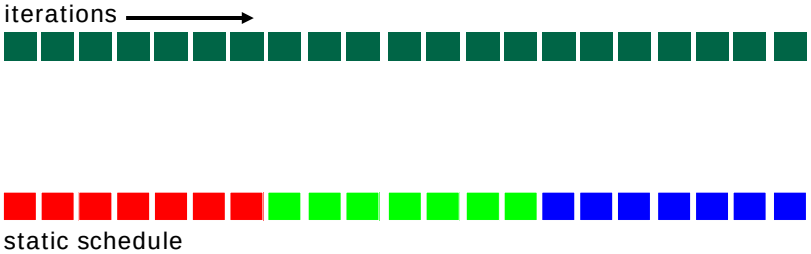
- Consider a loop…

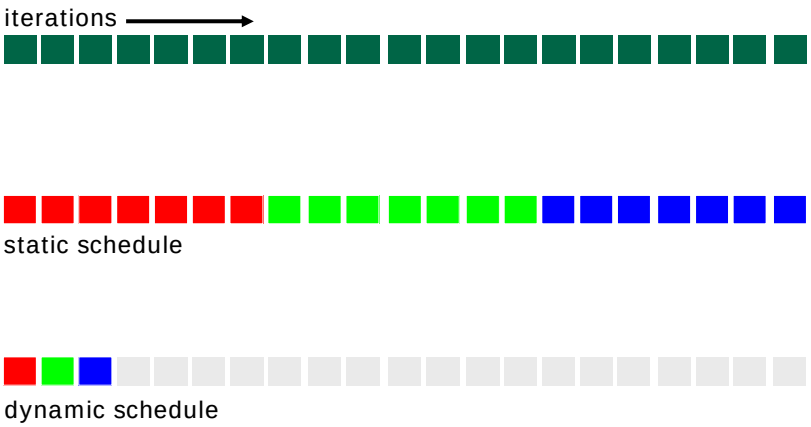
```
for (int i=0; i<N; i++)
    work(...);
```

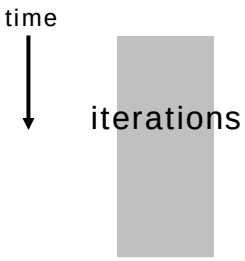
```
do i = 1, N
    call work(...)
end do
```

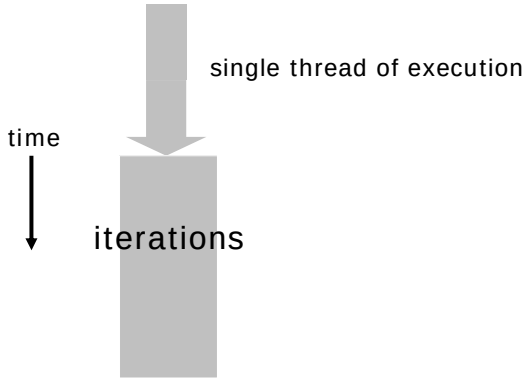
execute each block of iterations
on a separate processor

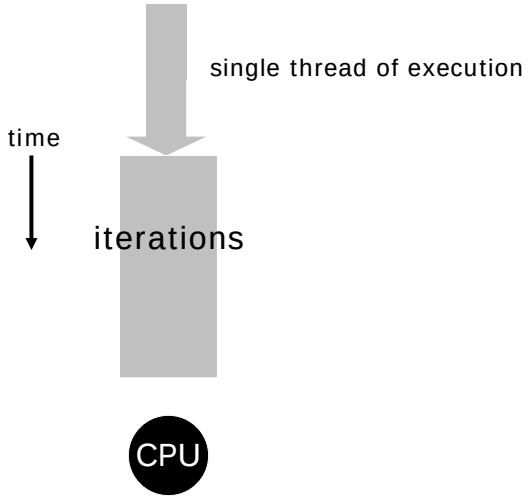


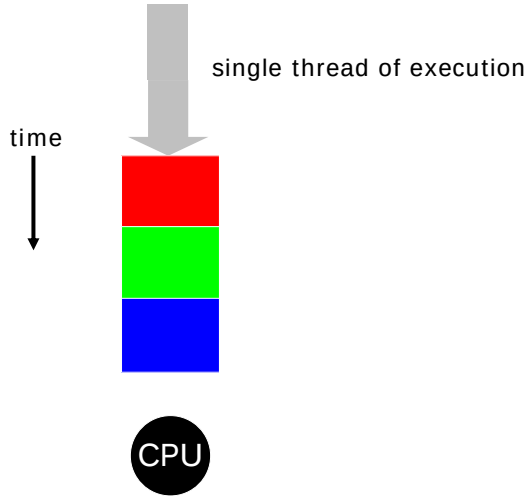


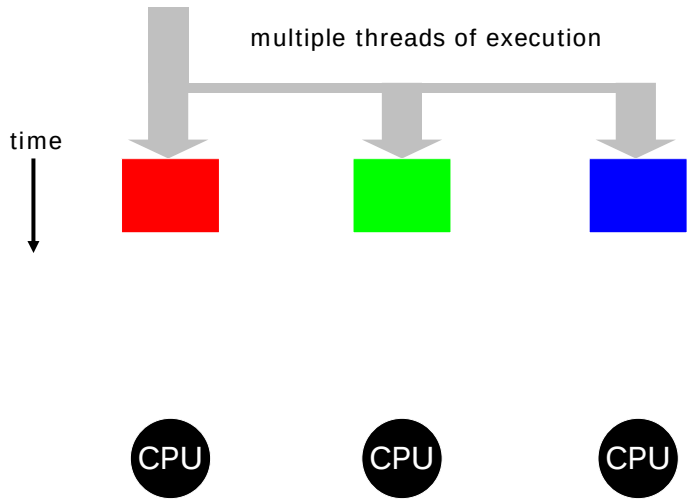


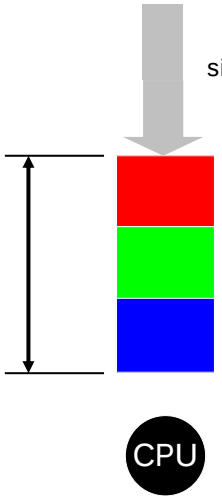


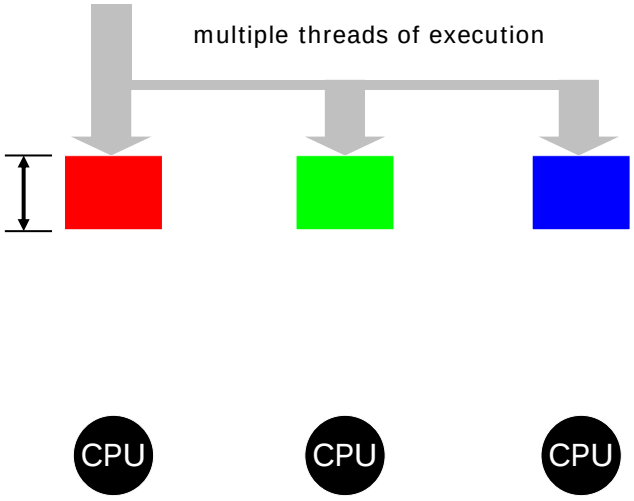


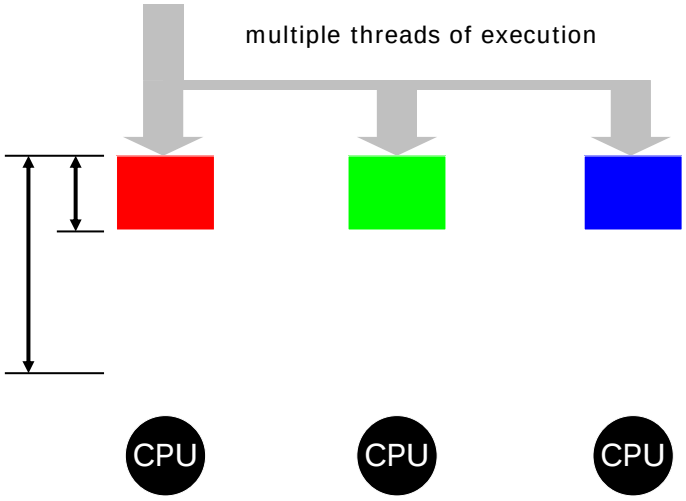












- Suitable WestGrid sites
  - **nexus**
    - Largest node has 256 CPUs and 256GB
  - **cortex**
    - Largest nodes have 64 CPUs and 256GB
  - **robson**
    glacier
    lattice
    matrix
    - At least two CPUs per node and 2GB
  - More multiprocessor/multicore nodes coming (WestGrid 2)

- How to create multiple threads of execution?

- Standard **POSIX threads**
- Create threads explicitly with Pthreads library
- Unavailable or non-standard (IBM) for Fortran
- Difficult to master

- Standard **OpenMP**
- Create threads implicitly
- Available for C/C++ and Fortran 77/90/95
- Deceptively easy to use

- The OpenMP way...

```
for (int i=0; i<N; i++)
    work(...);
```

```
do i = 1, N
    call work(...)
end do
```

- The OpenMP way…using **compiler directives**

```
#pragma omp parallel for
for (int i=0; i<N; i++)
    work(...);
```

```
!$omp parallel do
do i = 1, N
    call work(...)
end do
!$omp end parallel do
```


- The OpenMP way…using **compiler directives**

```
#pragma omp parallel for
for (int i=0; i<N; i++)
    work(...);
```

```
!$omp parallel do
do i = 1, N
    call work(...)
end do
!$omp end parallel do
```

- OpenMP also supports directives for parallelizing **sections** of code

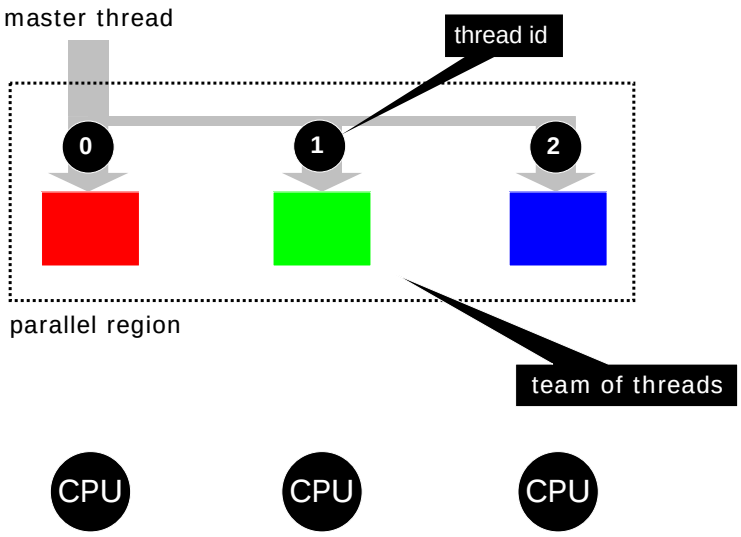
- Enable OpenMP at compile time (not standard)
  - SGI
    - Use the **-mp** switch
      f90 -mp source.f90
  - IBM
    - Use the reentrant compiler variants (with the **_r** suffix)
    - Use the **-qsmp=omp** switch
    - Fortran 77, include the **-qnosave** switch
      xlc_r -qsmp=omp source.c
  - Others
    - Portland **-mp**
    - Intel **-openmp**
    - GNU (>=4.2) **-fopenmp**

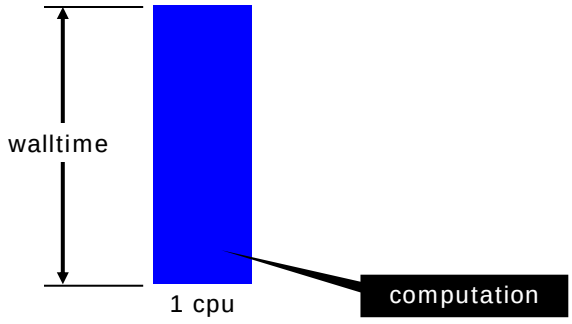
- Some compilers can automatically apply OpenMP directives to loops
  - SGI
    - **-apo**
  - IBM
    - **-qsmp=auto**
- Best to do it by hand
  - Parallelize one or two most time-consuming loops
  - Auto parallelization may miss valid loops

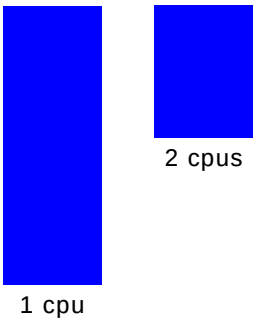
- Consider linking with a threaded math library
- No need to do any OpenMP in user code
  - SGI
    - **libscsl_mp.so**
  - IBM
    - **libesslsmp.a**
  - Others
    - Intel MKL (threaded by default)
    - ACML (mp versions)

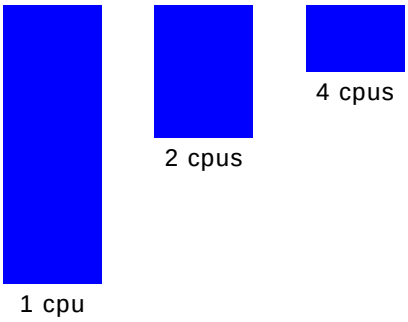
- Running an OpenMP program (standard)
  - Set **OMP_NUM_THREADS** environment variable
  - Execute program as usual

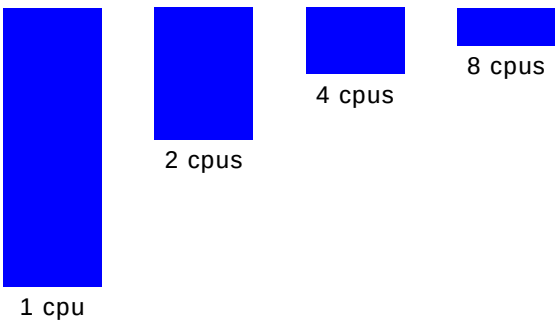
```
setenv OMP_NUM_THREADS 4
./a.out

setenv OMP_NUM_THREADS 8
./a.out
```







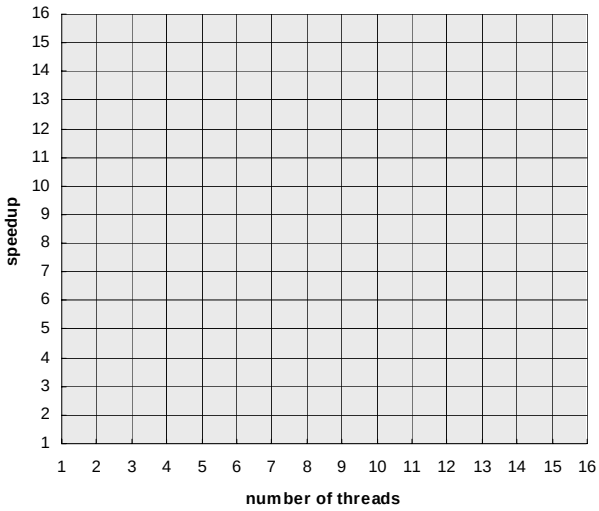


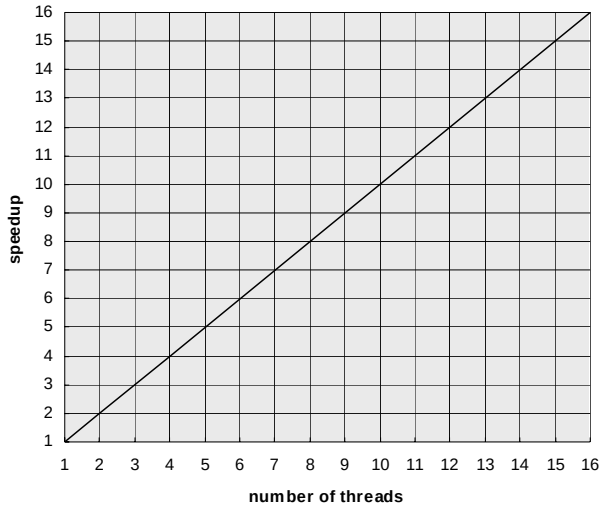


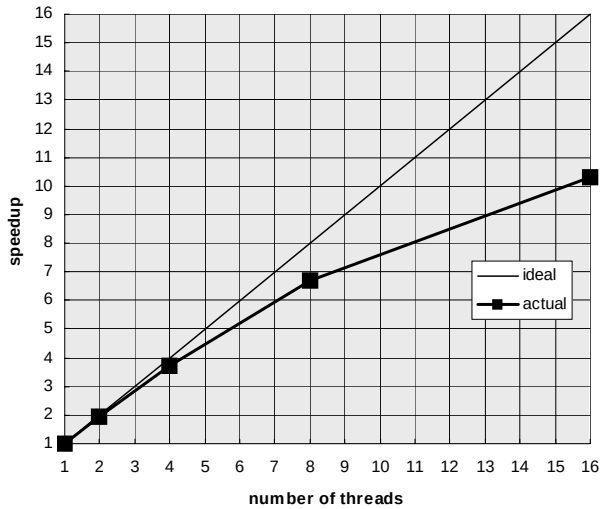
- In **theory**, OpenMP looks pretty good
- How about in **practice**…

- Scalability

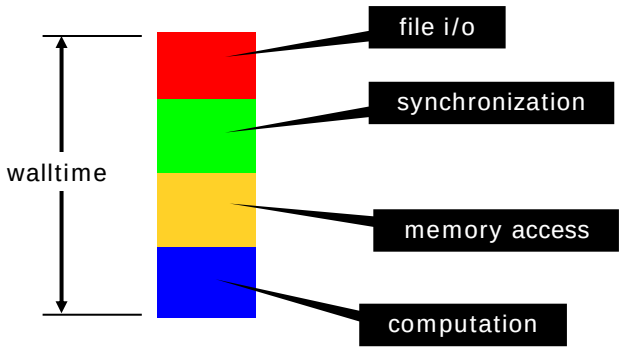
$$\text{speedup} \equiv \frac{\text{elapsed time of serial program}}{\text{elapsed time of parallel program}}$$

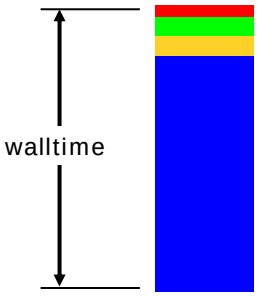




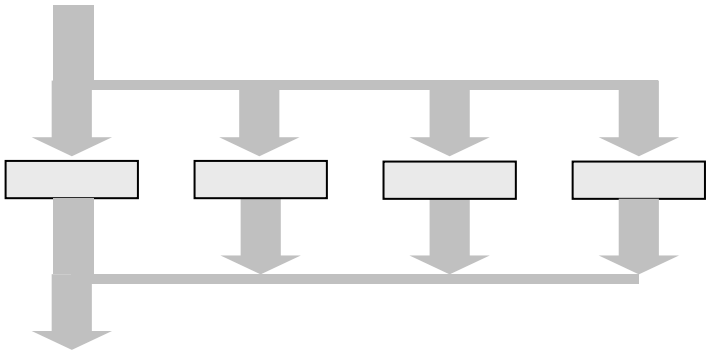


- In general, a parallel program's execution profile comprises
  - File I/O
  - Inter{process,thread} communication
  - Synchronization (barrier overhead)
  - Memory access
  - Computation

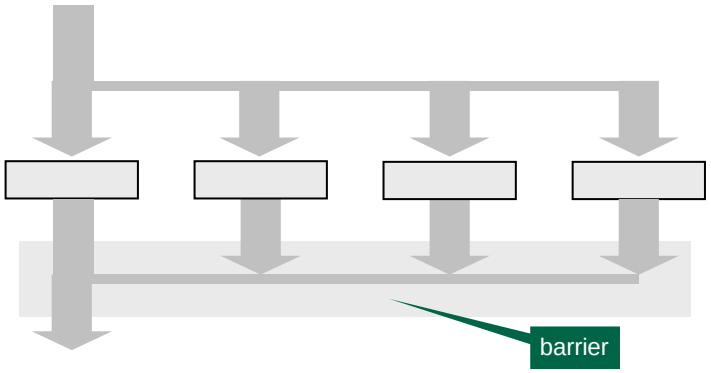




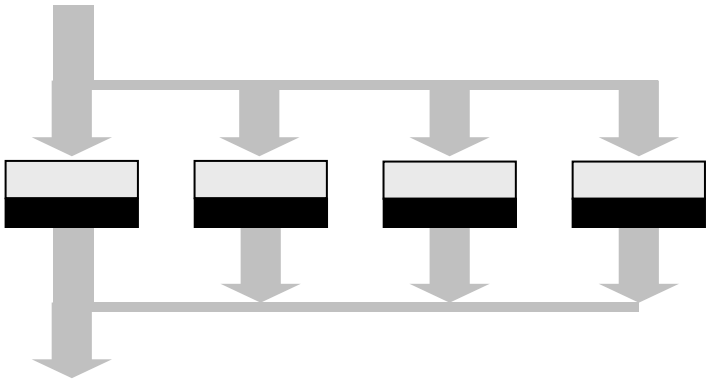
- Synchronization
  - Execution continues past a parallel region as soon as all the threads have finished



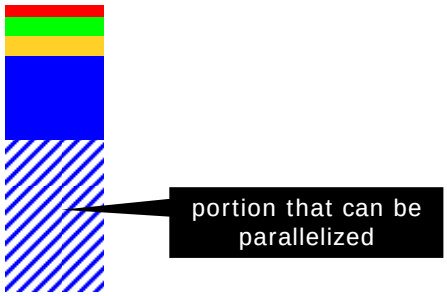
- Threads pause at a barrier to wait for all other threads to reach the barrier

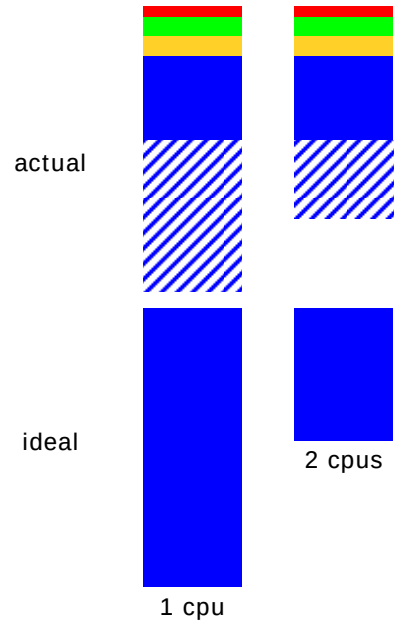


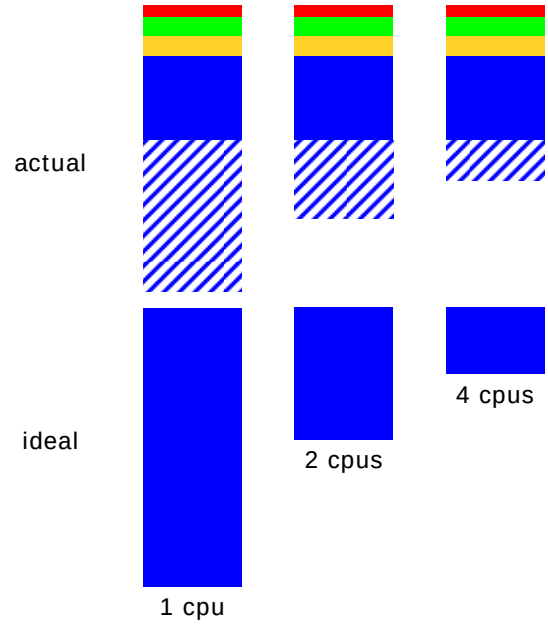
- The pause contributes overhead

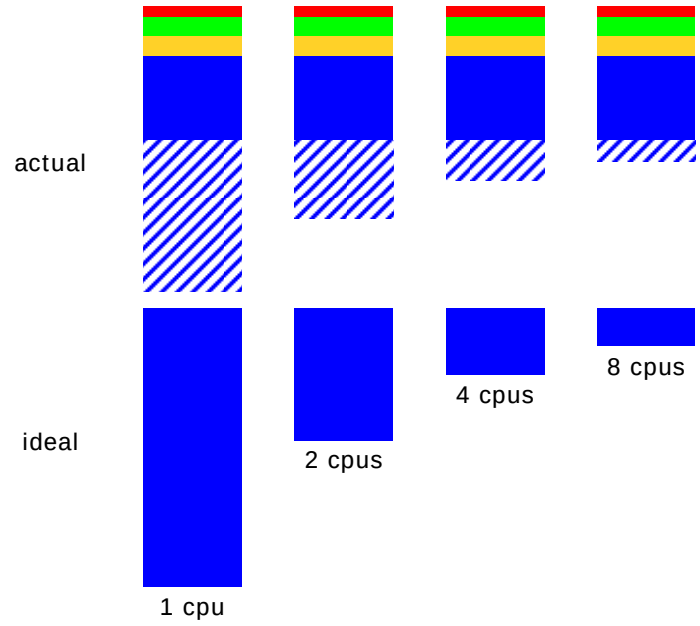


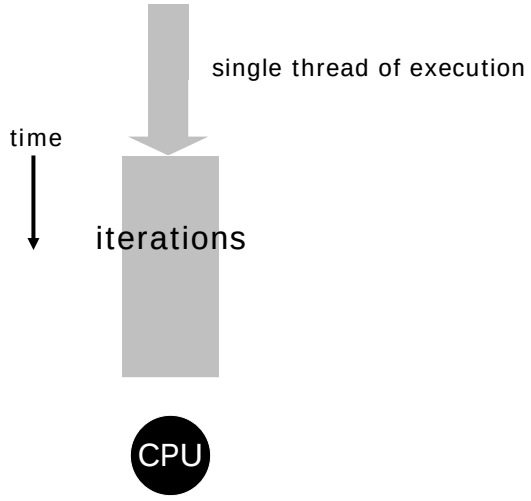
- Synchronization (barrier overhead)
  - Non-deterministic
  - Increases with number of threads
  - Increases with the number of parallel regions
  - To make OpenMP worthwhile, the amount of work in the parallel region should be "much larger" than the barrier overhead (on the order of "tens of millions of floating point operations," SGI)

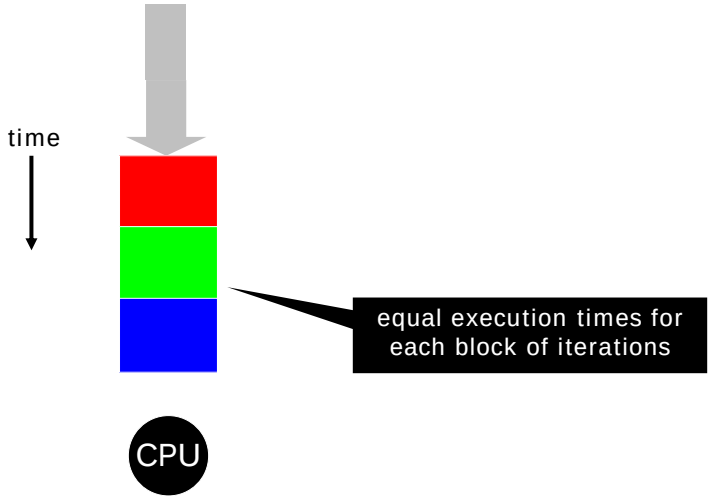


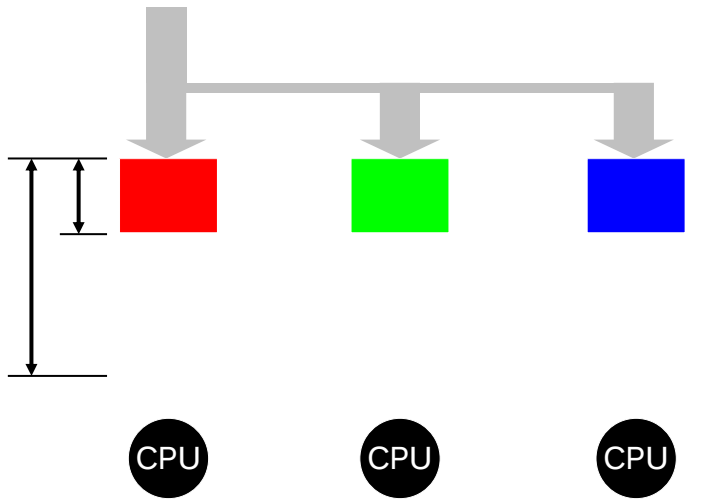


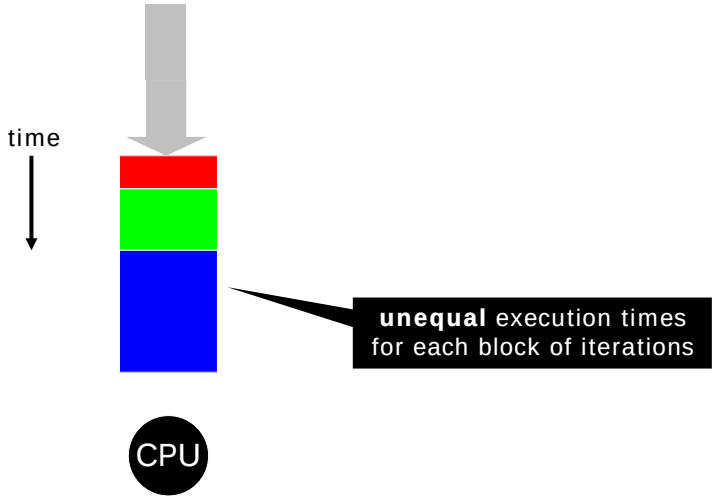


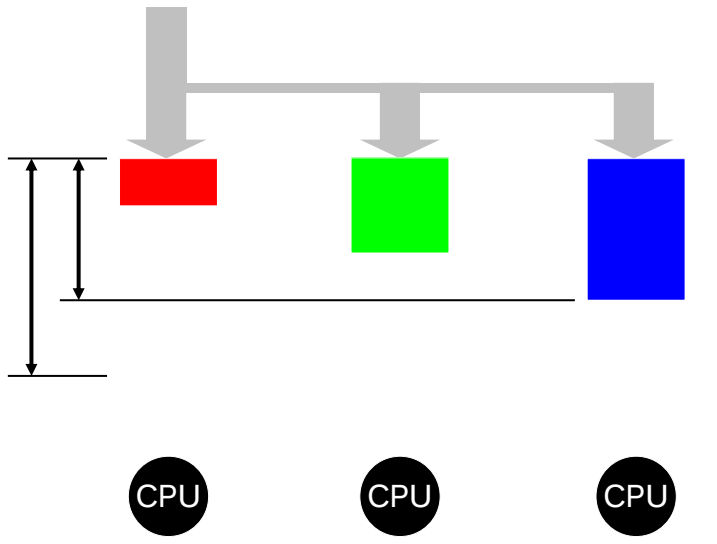










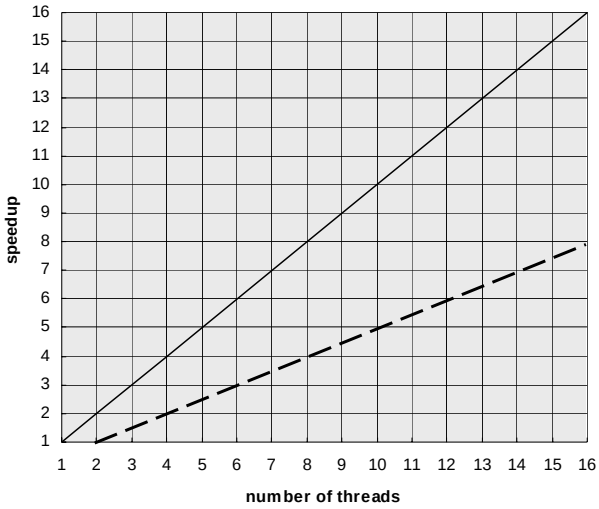


- There are many factors limiting scalability
- Not always possible to get good speedup
- What is good speedup?

- Recommendation (unofficial)…

Use as many threads as you need as long as the parallel efficiency is above 50%

- Ed's 50% rule



- OpenMP documentation
  - `http://www.openmp.org`
  - Click on "Specifications," get PDF Format files
  - Currently implemented version is 2.0
- WestGrid-specific information
  - `http://www.westgrid.ca/support/programming#parallel`

- Questions…

create delete