> By Jiahui Tang

Spec:

- OS: AWS VM Ubuntu 18.04
- Cluster: (x2.2large) * 2 instances
  - Latency:

    ```
    rtt min/avg/max/mdev = 0.349/0.479/2.914/0.239 ms
    ```
  - Bandwidth:

    ```
    Amazon says it’s 5 Gbps
    ```
- Compiler: gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

- More details below:

Network Latency:

```
ubuntu@ip-172-30-8-69:~$ ping node1
PING ip-172-30-8-84 (172.30.8.84) 56(84) bytes of data.
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=242 ttl=64 time=0.428 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=243 ttl=64 time=0.440 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=244 ttl=64 time=0.517 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=245 ttl=64 time=0.455 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=246 ttl=64 time=0.366 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=247 ttl=64 time=0.486 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=248 ttl=64 time=0.424 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=249 ttl=64 time=0.418 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=250 ttl=64 time=0.496 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=251 ttl=64 time=0.447 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=252 ttl=64 time=0.482 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=253 ttl=64 time=0.454 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=254 ttl=64 time=0.418 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=255 ttl=64 time=0.495 ms
64 bytes from ip-172-30-8-84 (172.30.8.84): icmp_seq=256 ttl=64 time=0.375 ms
```

```
--- ip-172-30-8-84 ping statistics ---
386 packets transmitted, 145 received, 62% packet loss, time 394111ms
rtt min/avg/max/mdev = 0.349/0.479/2.914/0.239 ms
```

Kernel Version:

```
ubuntu@ip-172-30-8-69:~$ uname -a
Linux ip-172-30-8-69 5.4.0-1038-aws #40~18.04.1-Ubuntu SMP Sat Feb 6 01:56:56 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

MPI Version:

```
ubuntu@ip-172-30-8-84:~$ mpiexec --version
HYDRA build details:
    Version:                                 3.3a2
    Release Date:                            Sun Nov 13 09:12:11 MST 2016
    CC:                              gcc   -Wl,-Bsymbolic-functions -Wl,-z,relro
    CXX:                             g++   -Wl,-Bsymbolic-functions -Wl,-z,relro
    F77:                             gfortran  -Wl,-Bsymbolic-functions -Wl,-z,relro
    F90:                             gfortran  -Wl,-Bsymbolic-functions -Wl,-z,relro
    Configure options:                       '--disable-option-checking' '--prefix=/usr' '--build=x86_64-linux-gnu' '--includedir=${prefix}/include' '--mandir=${prefix}/share/man' '--infodir=${prefix}/share/info' '--sysconfdir=/etc' '--lo
calstatedir=/var' '--disable-silent-rules' '--libdir=${prefix}/lib/x86_64-linux-gnu' '--libexecdir=${prefix}/lib/x86_64-linux-gnu' '--disable-maintainer-mode' '--disable-dependency-tracking' '--with-libfabric' '--enable-shared' '--enable-
fortran=all' '--disable-rpath' '--disable-wrapper-rpath' '--sysconfdir=/etc/mpich' '--libdir=/usr/lib/x86_64-linux-gnu' '--includedir=/usr/include/mpich' '--docdir=/usr/share/doc/mpich' '--with-hwloc-prefix=system' '--enable-checkpointing
' '--with-hydra-ckpointlib=blcr' 'CPPFLAGS= -Wdate-time -D_FORTIFY_SOURCE=2 -I/build/mpich-O9at2o/mpich-3.3~a2/src/mpl/include -I/build/mpich-O9at2o/mpich-3.3~a2/src/mpl/include -I/build/mpich-O9at2o/mpich-3.3~a2/src/openpa/src -I/build/m
pich-O9at2o/mpich-3.3~a2/src/openpa/src -D_REENTRANT -I/build/mpich-O9at2o/mpich-3.3~a2/src/mpi/romio/include' 'CFLAGS= -g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong -Wformat -Werror=format-security
 -O2' 'CXXFLAGS= -g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong -Wformat -Werror=format-security -O2' 'FFLAGS= -g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong -O
2' 'FCFLAGS= -g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong -O2' 'build_alias=x86_64-linux-gnu' 'MPICHLIB_CFLAGS=-g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong
-Wformat -Werror=format-security' 'MPICHLIB_CPPFLAGS=-Wdate-time -D_FORTIFY_SOURCE=2' 'MPICHLIB_CXXFLAGS=-g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong -Wformat -Werror=format-security' 'MPICHLIB_FFL
AGS=-g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong' 'MPICHLIB_FCFLAGS=-g -O2 -fdebug-prefix-map=/build/mpich-O9at2o/mpich-3.3~a2=. -fstack-protector-strong' 'LDFLAGS=-Wl,-Bsymbolic-functions -Wl,-z,r
elro' 'FC=gfortran' 'F77=gfortran' 'MPILIBNAME=mpich' '--cache-file=/dev/null' '--srcdir=.' 'CC=gcc' 'LIBS=' 'MPLLIBNAME=mpl'
    Process Manager:                         pmi
    Launchers available:                     ssh rsh fork slurm ll lsf sge manual persist
    Topology libraries available:            hwloc
    Resource management kernels available:   user slurm ll lsf sge pbs cobalt
    Checkpointing libraries available:       blcr
    Demux engines available:                 poll select
ubuntu@ip-172-30-8-84:~$
```

CPU Info:

```
ubuntu@ip-172-30-8-69:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 79
model name      : Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
stepping        : 1
microcode       : 0xb000038
cpu MHz         : 2300.073
cache size      : 46080 KB
physical id     : 0
siblings        : 8
core id         : 0
cpu cores       : 8
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx pdpe1gb rdtscp lm constant_tsc rep_good nopl xtopology cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2
 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm cpuid_fault invpcid_single pti fsgsbase bmi1 avx2 smep bmi2 erms invpcid xsaveopt
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds swapgs itlb_multihit
bogomips        : 4600.10
clflush size    : 64
cache_alignment : 64
address sizes   : 46 bits physical, 48 bits virtual
power management:
```

```
mpiuser@ip-172-30-8-69:~/cloud$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              8
On-line CPU(s) list: 0-7
Thread(s) per core:  1
Core(s) per socket:  8
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               79
Model name:          Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
Stepping:            1
CPU MHz:             2299.865
BogoMIPS:            4600.16
Hypervisor vendor:   Xen
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            46080K
NUMA node0 CPU(s):   0-7
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmo
4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor l
```

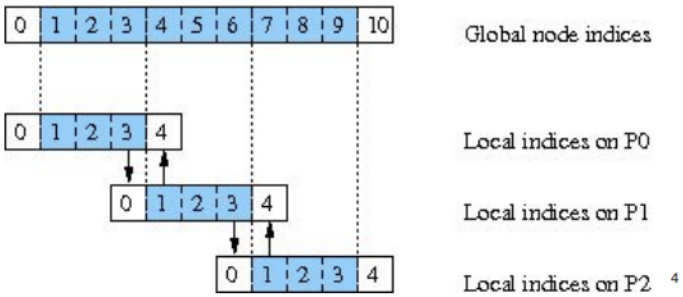
Compiler:

```
ubuntu@ip-172-30-8-69:~$ mpicc --version
gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

5. Distributed-Memory Parallel Processing (35 points)

5.1. Parallelization of Basic Code (20 points)

The goal of this exercise[3] is to familiarize you with MPI and make your first parallel code. Develop a MPI version of the jacobi code used in section 1.1 by completing `jacobi1d_mpi.c`. The structure of `jacobi1d_mpi.c` is somewhat more complicated. In this code, no processor holds the entire solution vector; instead, each processor has a local piece of the vector, and these pieces overlap slightly at the ends. No variable is directly updated by more than one processor; the overlap is simply to accommodate *ghost cells*. As a concrete example, suppose we discretize the interval [0,1] into 10 cells of size 0.1. Then there are 11 points in our mesh, including the endpoints subject to boundary conditions. If we partition these points among three processors, we have the following picture:



[4]

In this picture, the active variables are colored blue and the white boxes at the end correspond to storage for ghost cells or boundary data. For example, consider the storage on the second processor (P1). Entries 1, 2, and 3 in this processor's local vector correspond to entries 4, 5, and 6 in the global indexing of variables. However, in order to update these entries, P1 must know about the entries 3 and 7 in the global vector. P1 has storage for these variables in 0 and 4 of its local vector; but rather than computing these variables, P1 must be sent the information from the neighboring processor. Thus, at each step,

- P0 sends entry 3 of its local vector to P1 and receives entry 4;
- P1 sends entry 1 of its local vector to P0 and receives entry 0;
- P1 sends entry 3 of its local vector to P2 and receives entry 4;
- P2 sends entry 1 of its local vector to P1 and receives entry 0

---

[3] http://www.cs.cornell.edu/~bindel/class/cs5220-f11/hw2.html

[4] In fact, the provided skeleton code splits global indices in a particular way and - as some of you might have noticed - the picture for your execution of *n=10* will look more like this: https://harvard-iacs.github.io/2021-CS205/homeworks/HWB/guide/indices_table.png.
This does not result in a perfectly even distribution, but will be reasonably efficient when the size of n is sufficiently large. Your portion of the code will also not be affected, so you can disregard this particularity.

In `jacobi1d_mpi.c`, the code that implements this exchange of ghost cell information is left out. You should fill this code in. Your code should be correct for any number of processors, not just two or three. The solution uses `MPI_Send`, and `MPI_Recv` in two phases: one that sends data to the processor to the left, and one that sends data to the processor to the right. However, if you would prefer to use a different organization, you are welcome to do so.

You should do the following experiments:

1. Run your code with 1 to 4 tasks on one node with `ncells` equal to 10^8 and 100 steps.
2. Run your code with 8 tasks on two nodes with `ncells` equal to 10^k, for k = 5, 6, 7, and 8 and 100 steps. Plot the ratio of your parallel run times to the serial code run time as a function of k. Calculate the Speed-up and explain why it increases or decreases with the problem size. Make sure to change your instance type to t2.2xlarge when running k = 8.
3. Run your codes with `ncells` of 100,000,000 and 100 steps with 2-8 tasks on two nodes and produce a speedup plot. Explain why it increases or decreases with the number of cores.
4. Compare the performance results with those obtained with OpenMP.

You should check the correctness of your implementation by comparing the solution output files to the solution output files from the serial code.

Submission

- `P51.pdf`: with replicability information (see Submission note above), Explain your results in terms of a simple performance model that takes into account the time spent on both computation (which is proportional to the number of points per processor) and communication/synchronization
- `P51.c`: Complete source code. In your source code, please insert comments to explain what you are using each of the MPI functions for

(5.1.1)

With `ncells = 10^8 , step=100 , 1 node`

Unit: Seconds

Number of Processes	1	2	3	4
One Node	48.328	24.0397	16.0932	12.0808

Discussion:

On one node, the performance increases and time decreases as number of tasks increase. This is similar to what we discussed in question 4, as more processor task per node allow more concurrency when shared same memory, that could simultaneously execute statements in the parallel region.

Reference:

```
mpiuser@ip-172-30-8-69:~$ mpicc -DUSE_CLOCK p51.c timing.c -o p51
mpiuser@ip-172-30-8-69:~$ mpirun -np 1 ./p51 100000000 100
Processes: 1
n: 100000000
nsteps: 100
Elapsed time: 48.328 s
mpiuser@ip-172-30-8-69:~$ mpirun -np 2 ./p51 100000000 100
Processes: 2
n: 100000000
nsteps: 100
Elapsed time: 24.0397 s
mpiuser@ip-172-30-8-69:~$ mpirun -np 3 ./p51 100000000 100
Processes: 3
n: 100000000
nsteps: 100
Elapsed time: 16.0932 s
mpiuser@ip-172-30-8-69:~$ mpirun -np 4 ./p51 100000000 100
Processes: 4
n: 100000000
nsteps: 100
Elapsed time: 12.0808 s
```

(5.1.2). Table for improvements in elapsed execution time

(*unit: seconds*)

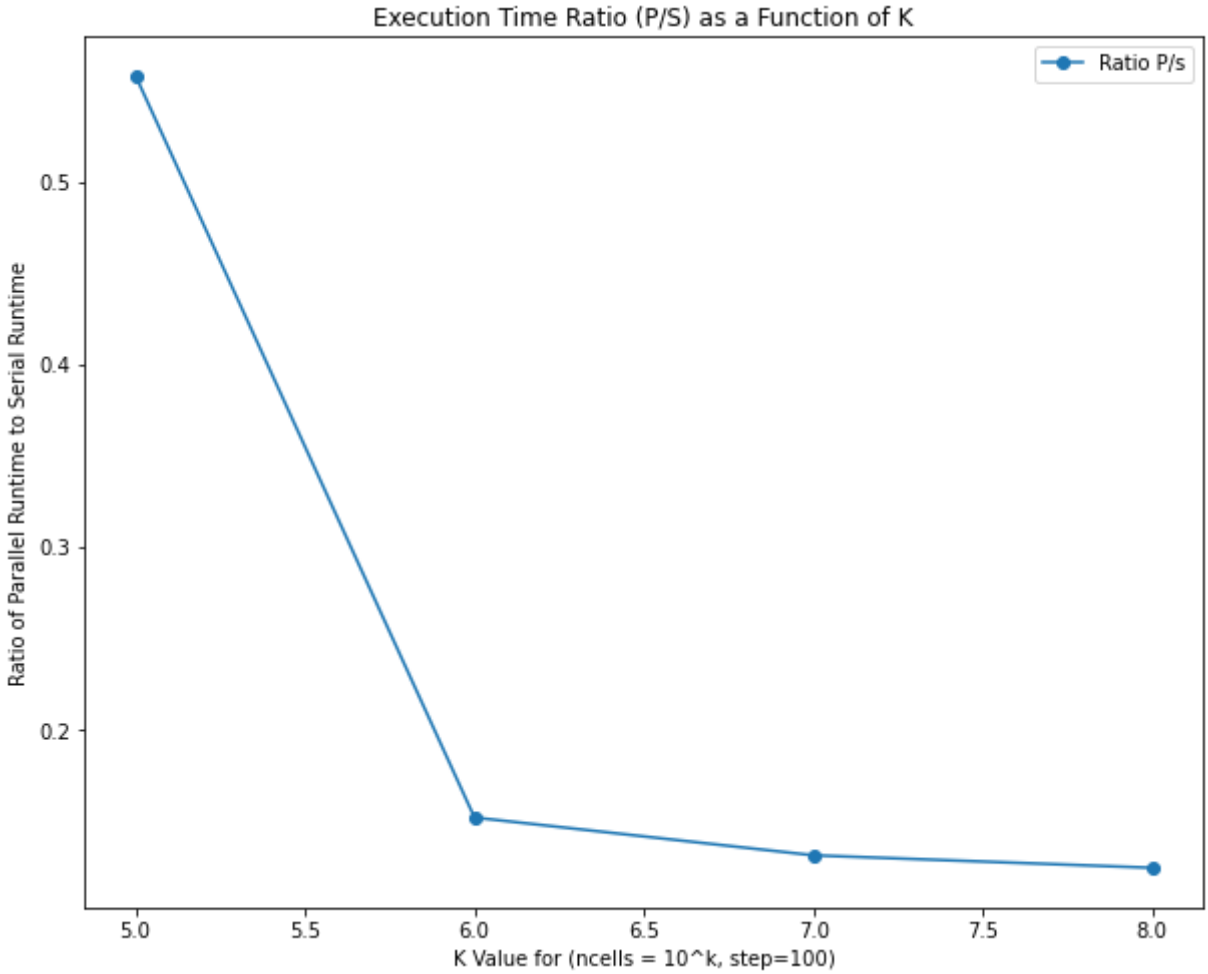
With `ncells = 10^k , step=100 , -np 8`

For serial code runtime, it has 8 different tasks and 8 elapsed time, I will take an average time for these 8 tasks. For example, for serial code, when running on 8 task, it shows the below 8 elapsed time in attached screenshot. I would take an average and put `0.04383175` for `k=5 serial code elpased time`

	k=5 Elpased Time	k=6 Elpased Time	k=7 Elpased Time	k=8 Elpased Time
Parallel Code Runtime	0.0244453	0.0734067	0.631309	6.07382
Serial Code Runtime	0.04383175	0.48234	4.79949	48.70995
Ratio P/S	0.55788	0.15219	0.13153	0.12469
Speedup S/P	1.79249	6.57078	7.60244	8.01966

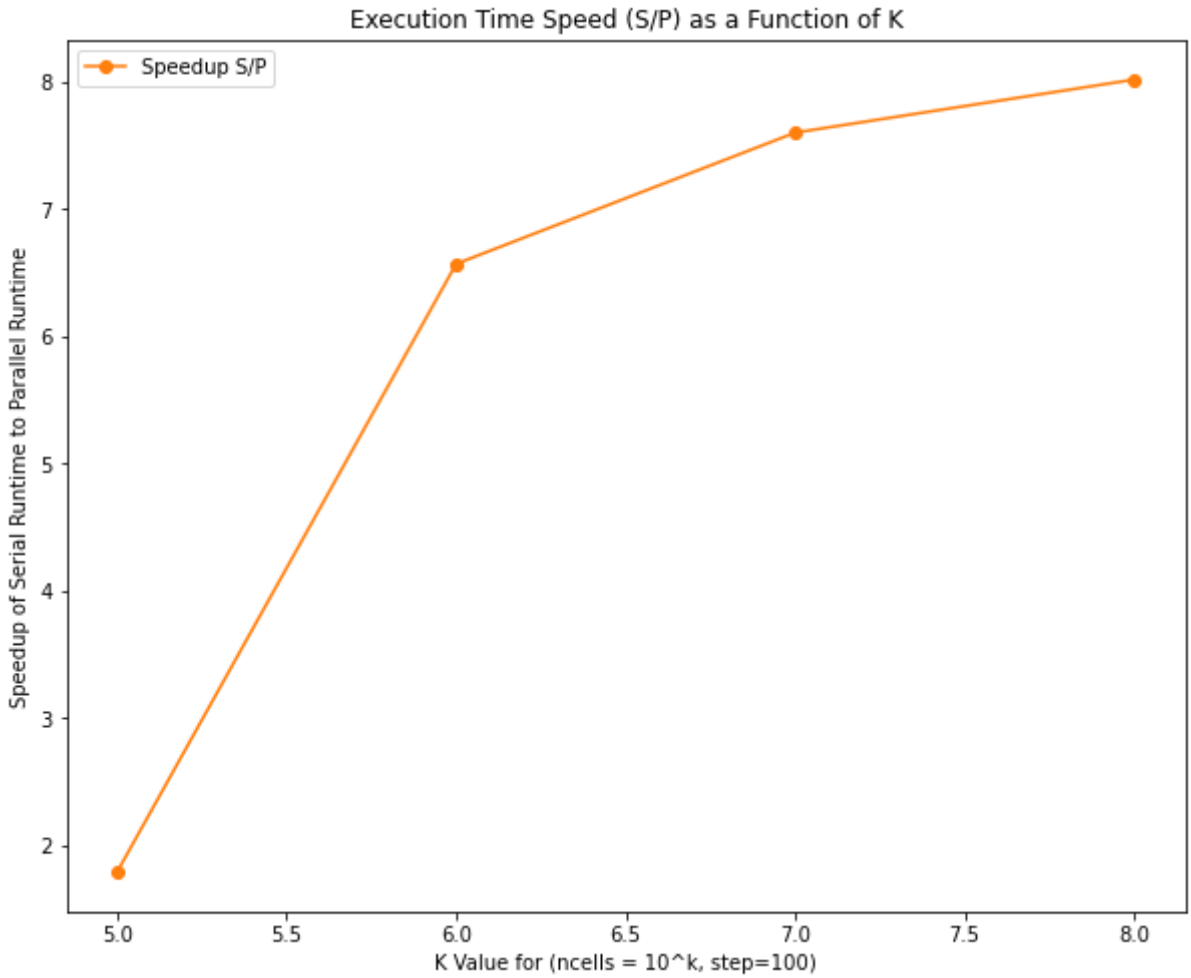
In [3]:

```
import matplotlib.pyplot as plt

k = [5,6,7,8]
ratio = [0.55788,0.15219,0.13153,0.12469]
speedup = [1.79249,6.57078,7.60244,8.01966]

plt.figure(figsize = (10,8))
plt.plot(k, ratio, '-ob', label = "Ratio P/s", color="tab:blue")
#plt.plot(k, speedup, '-ob', label = "Speedup S/P",color="tab:orange")

plt.xlabel('K Value for (ncells = 10^k, step=100)')
plt.ylabel('Ratio of Parallel Runtime to Serial Runtime')
plt.title('Execution Time Ratio (P/S) as a Function of K')
plt.legend()
plt.show()
```



In [2]:

```
plt.figure(figsize = (10,8))
#plt.plot(k, ratio, '-ob', label = "Ratio P/s", color="tab:blue")
plt.plot(k, speedup, '-ob', label = "Speedup S/P",color="tab:orange")

plt.xlabel('K Value for (ncells = 10^k, step=100)')
plt.ylabel('Speedup of Serial Runtime to Parallel Runtime')
plt.title('Execution Time Speedup (S/P) as a Function of K')
plt.legend()
plt.show()
```



Discussion:

> The speedup increases as K value increases, it shows when problem size increase, the speedup also increases. However it also suffer from a diminishing return with slower incremental and smaller marginal speedup when problem size gets larger. It eventually approaches 8, as there are 8 parallel tasks and the theortical maximum speedup is 8.
>
> Speedup increase with problem size because as we fixed number of thread, the overhead for synchronization and communication is roughly the same for all problem sizes. When K increases, the overheads will thus take a smaller and neglible proportion of overall runtime. Total runtime adds up the parallel time and overhead time, thus it will result in a higher and higher speedup.
>
> It also suffer from a diminishing marginal increments because as problem size goes up, the overhead time become insignificant. The speedup thus gradually approaches real speedup.
>
> Also there's diminishing marginal incremental in speedup rate when problem size goes up, as `granularity= computation/communication ratio` , when fixing number of cores, changing problem size changes the computation portion of granularity. The granularity increased to a higher ratio, became `coarse-grained` , where relatively large amounts of computational work are done between communication/synchronization events. Thus, gradually, the load imbalance overhead may dominates overhead, and also lead to higher execution time. It slows

> down the acceleration of speedup, making the marginal increase became smaller as problem size goes up.
>
> Finally, the speedup follows Amdahl's Law, is defined by the fraction of code (c) that can be parallelized, and bounded by sequential code, even a small percentage of sequential code can greatly limit potential speedup. Because it has overhead, thus the speedup never reaches theory speedup but were approaching limit (here may also suffer from a bit of floating point errors).

Reference:

- MPI Parallel Code:

```
ubuntu@ip-172-30-8-69:~$ su - mpiuser
Password:
mpiuser@ip-172-30-8-69:~$ export MPICH_PORT_RANGE=10000:10100
mpiuser@ip-172-30-8-69:~$ export MPIR_CVAR_CH3_PORT_RANGE=10000:10100
mpiuser@ip-172-30-8-69:~$ ls
 cloud  p51  p51.c  timing.c  timing.h
mpiuser@ip-172-30-8-69:~$ cd cloud
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./p51 100000 100
 Processes: 8
 n: 100000
 nsteps: 100
 Elapsed time: 0.0244453 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./p51 1000000 100
 Processes: 8
 n: 1000000
 nsteps: 100
 Elapsed time: 0.0734067 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./p51 10000000 100
 Processes: 8
 n: 10000000
 nsteps: 100
 Elapsed time: 0.631309 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./p51 100000000 100
 Processes: 8
 n: 100000000
 nsteps: 100
 Elapsed time: 6.07382 s
mpiuser@ip-172-30-8-69:~/cloud$
```

- Serial Code

```
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./jacobi1d 100000 100
n: 100000
nsteps: 100
Elapsed time: 0.043732 s
n: 100000
nsteps: 100
Elapsed time: 0.043712 s
n: 100000
nsteps: 100
Elapsed time: 0.043673 s
n: 100000
nsteps: 100
Elapsed time: 0.043638 s
n: 100000
nsteps: 100
Elapsed time: 0.043678 s
n: 100000
nsteps: 100
Elapsed time: 0.0437 s
n: 100000
nsteps: 100
Elapsed time: 0.044072 s
n: 100000
nsteps: 100
Elapsed time: 0.044449 s
```

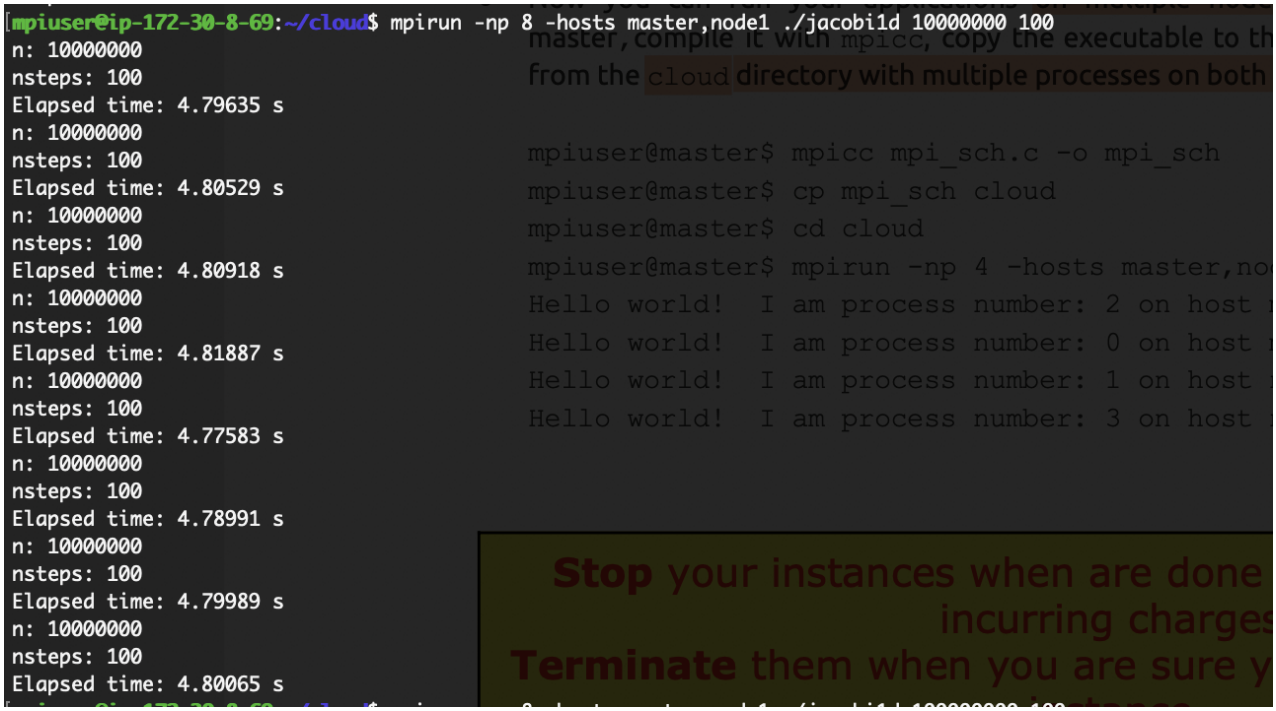
```
mpiuser@master$ mpicc mpi_sch.c -o mpi_
mpiuser@master$ cp mpi_sch cloud
mpiuser@master$ cd cloud
mpiuser@master$ mpirun -np 4 -hosts mas
Hello world!  I am process number: 2 on
Hello world!  I am process number: 0 on
Hello world!  I am process number: 1 on
Hello world!  I am process number: 3 on
```

Stop your instances when are
incurring ch
Terminate them when you are s
instanc

```
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./jacobi1d 1000000 100
n: 1000000
nsteps: 100
Elapsed time: 0.481575 s
n: 1000000
nsteps: 100
Elapsed time: 0.482105 s
n: 1000000
nsteps: 100
Elapsed time: 0.482177 s
n: 1000000
nsteps: 100
Elapsed time: 0.486258 s
n: 1000000
nsteps: 100
Elapsed time: 0.479203 s
n: 1000000
nsteps: 100
Elapsed time: 0.48201 s
n: 1000000
nsteps: 100
Elapsed time: 0.482349 s
n: 1000000
nsteps: 100
Elapsed time: 0.483043 s
```

```
mpiuser@master$ mpicc mpi_sch.c -o mpi_s
mpiuser@master$ cp mpi_sch cloud
mpiuser@master$ cd cloud
mpiuser@master$ mpirun -np 4 -hosts mast
Hello world!  I am process number: 2 on
Hello world!  I am process number: 0 on
Hello world!  I am process number: 1 on
Hello world!  I am process number: 3 on
```

Stop your instances when are d
incurring cha
Terminate them when you are su
instance



(5.1.3). Table for improvements in elapsed execution time on two nodes with different tasks

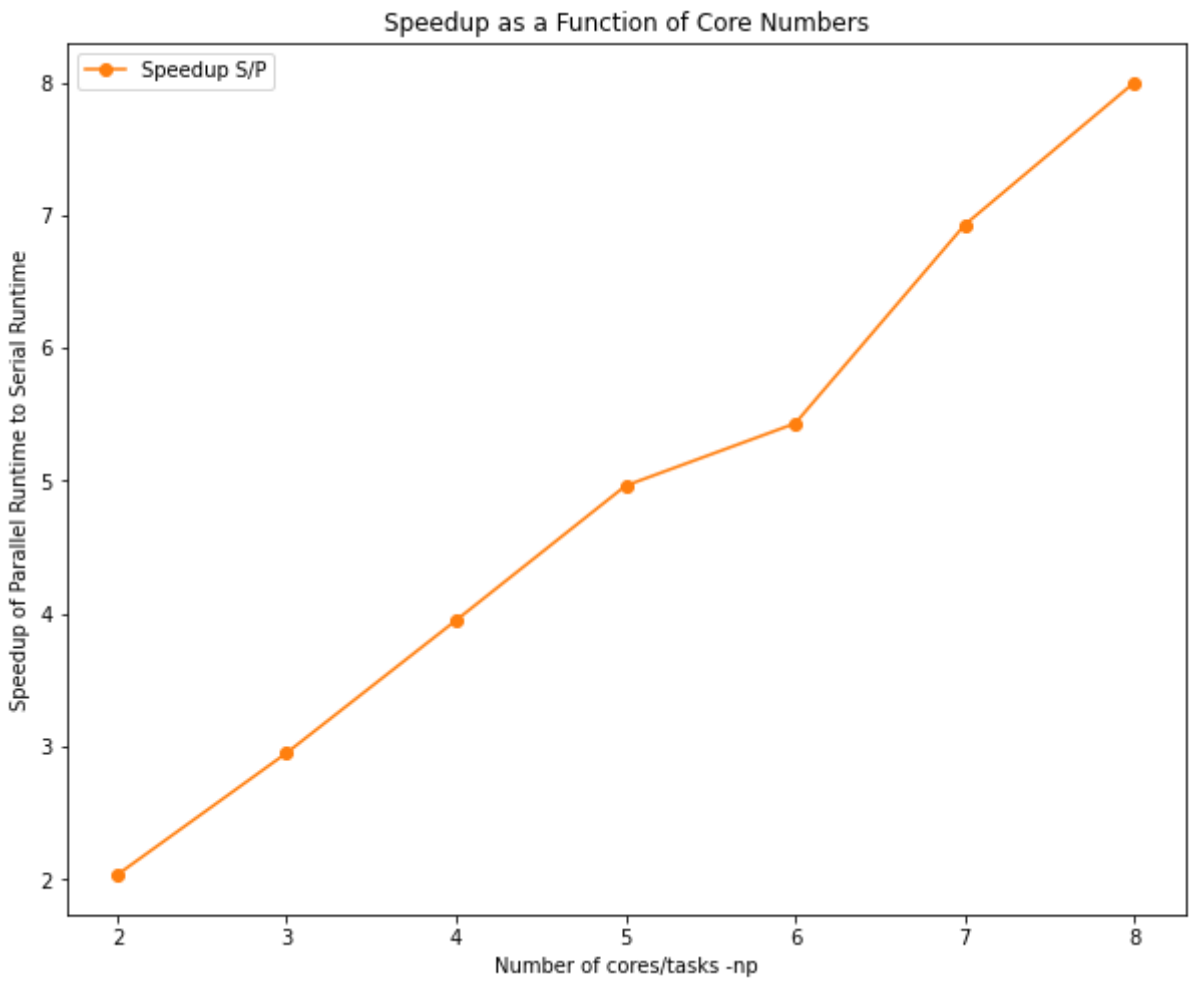
(*unit: seconds*)

With `ncells = 10^8`, `step=100`, `2 nodes`

np (number of tasks)	-np 2	-np 3	-np 4	-np 5	-np 6	-np 7	-np 8
Parallel Code Runtime	23.6753	16.1857	12.0332	9.67444	8.92306	6.99572	6.08274
Sequential Code Runtime	48.19455	47.8041	47.55055	48.0219	48.47373	48.45667	48.6451
Ratio P/S	0.49124	0.33858	0.25306	0.20145	0.18408	0.14437	0.12504
Speedup S/P	2.03565	2.95348	3.9516	4.96379	5.43241	6.92662	7.99723

In [8]:
```
import matplotlib.pyplot as plt
```

```
k = [2,3,4,5,6,7,8]
ratio = [0.49124,0.33858,0.25306,0.20145,0.18408,0.14437,0.12504]
speedup = [2.03565,2.95348,3.9516,4.96379,5.43241,6.92662,7.99723]

plt.figure(figsize = (10,8))
#plt.plot(k, ratio, '-ob', label = "Ratio P/s", color="tab:blue")
plt.plot(k, speedup, '-ob', label = "Speedup S/P",color="tab:orange")

plt.xlabel("Number of cores/tasks -np" )
plt.ylabel('Speedup of Parallel Runtime to Serial Runtime')
plt.title('Speedup as a Function of Core Numbers')
plt.legend()
plt.show()
```



Discussion:

> With all the other criteria hold same, as the number of cores/MPI tasks increase, the speedup value also increases linearly with it. It makes sense because when we are using MPI to distribute parallelized code into `n cores`, the maximum theortical speed up is thus `n`. More details is discussed below.
>
> P.S: The turning point at `6` here might be due to floating point errors, the overall trend is linear.

5.1.4.

Discussion:

In **OpenMP**, which is shared memory parallel processing, the speedup become slower when core increases, the maximum speedup is around `3.4`.

However, in **MPI** with distributed memory and multiple cores, the maximum speedup increases linearly with number of cores, and could reach theortical value of `8`.

While we shouldn't compare the absolute value of elapsed time, because looking at serial runtime OpenMp version and MPI version display very different average time for all threads. One is around `48 sec` and the other is around `24 secs`.

Instead, I will compare speed up of these two parallel processing.

For **MPI**, it uses 8 threads in 8 cores (as there are double node in a cluster), it is `1 thread/core`, meaning all threads are fully used concurrently. Thus, when number of cores increases, the linear increase of speedup is as expected.

For **OpenMP**, it uses 8 threads in 4 cores (Intel, single node), thus when `core > 4`, the overhead is increasing and obviously slows down speedup increment.

It is mainly because CPU cache is allocated core-wise. And there will there be different memory distribution methods in OpenMP and MPI. Thus if there are two threads in one core, they will use the same memory cache and thus produce more communication and sychronization overhead. It results in slowing down of speedup and also couldn't reach theortical speedup of `4x` in OpenMP. However, in MPI, it is `1 thread/core`, and there will not be this problem, it could thus reach speed up of theortical `8x`. Also for MPI, there is overhead for creating process for one time, where in OpenMP, the overhead depends on implementation threads, and can be created or joined for particularly task which add more overhead. Here for this problem, MPI have a better speedup.

(4.1.2). Table for improvements in elapsed execution time

(*unit: seconds*)

With `ncells = 10^8`, `step=100`, `OMP_NUM_THREADS=8`, `-Ofast`

OMP_NUM_THREADS	2 cores	3 cores	4 cores	5 cores	6 cores	7 cores	8 cores
Parallel Code Runtime	14.742	10.665	8.868	8.066	7.614	7.363	7.238
Sequential Code Runtime	25.0803	24.9811	25.0209	24.8697	24.9831	24.8953	24.7347
Ratio P/S	0.5878	0.4269	0.3544	0.3243	0.3048	0.2957	0.2926
Speedup S/P	1.7013	2.3423	2.8215	3.0833	3.2812	3.3811	3.4173

Reference:

- Parallel:

```
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./p51 100000000 100
Processes: 8
n: 100000000
nsteps: 100
Elapsed time: 6.08274 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 7 -hosts master,node1 ./p51 100000000 100
Processes: 7
n: 100000000
nsteps: 100
Elapsed time: 6.99572 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 6 -hosts master,node1 ./p51 100000000 100
Processes: 6
n: 100000000
nsteps: 100
Elapsed time: 8.92306 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 5 -hosts master,node1 ./p51 100000000 100
Processes: 5
n: 100000000
nsteps: 100
Elapsed time: 9.67444 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 4 -hosts master,node1 ./p51 100000000 100
Processes: 4
n: 100000000
nsteps: 100
Elapsed time: 12.0332 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 3 -hosts master,node1 ./p51 100000000 100
Processes: 3
n: 100000000
nsteps: 100
Elapsed time: 16.1857 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 2 -hosts master,node1 ./p51 100000000 100
Processes: 2
n: 100000000
nsteps: 100
Elapsed time: 23.6753 s
```

- Serial:

```
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 2 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 48.4575 s
n: 100000000
nsteps: 100
Elapsed time: 47.9316 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 3 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 47.4405 s
n: 100000000
nsteps: 100
Elapsed time: 47.5692 s
n: 100000000
nsteps: 100
Elapsed time: 48.4026 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 4 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 47.4278 s
n: 100000000
nsteps: 100
Elapsed time: 47.5573 s
n: 100000000
nsteps: 100
Elapsed time: 47.5211 s
n: 100000000
nsteps: 100
Elapsed time: 47.696 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 5 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 47.3352 s
n: 100000000
nsteps: 100
Elapsed time: 48.1289 s
n: 100000000
nsteps: 100
Elapsed time: 47.7299 s
n: 100000000
nsteps: 100
Elapsed time: 48.3616 s
n: 100000000
nsteps: 100
Elapsed time: 48.5539 s
```

```
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 6 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 47.8831 s
n: 100000000
nsteps: 100
Elapsed time: 48.0011 s
n: 100000000
nsteps: 100
Elapsed time: 48.033 s
n: 100000000
nsteps: 100
Elapsed time: 48.8484 s
n: 100000000
nsteps: 100
Elapsed time: 49.0313 s
n: 100000000
nsteps: 100
Elapsed time: 49.0455 s
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 7 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 48.0715 s
n: 100000000
nsteps: 100
Elapsed time: 48.0817 s
n: 100000000
nsteps: 100
Elapsed time: 48.2545 s
n: 100000000
nsteps: 100
Elapsed time: 48.4328 s
n: 100000000
nsteps: 100
Elapsed time: 48.6857 s
n: 100000000
nsteps: 100
Elapsed time: 48.8315 s
n: 100000000
nsteps: 100
Elapsed time: 48.839 s
```

```
mpiuser@ip-172-30-8-69:~/cloud$ mpirun -np 8 -hosts master,node1 ./jacobi1d 100000000 100
n: 100000000
nsteps: 100
Elapsed time: 47.827 s
n: 100000000
nsteps: 100
Elapsed time: 48.046 s
n: 100000000
nsteps: 100
Elapsed time: 48.1666 s
n: 100000000
nsteps: 100
Elapsed time: 48.3471 s
n: 100000000
nsteps: 100
Elapsed time: 49.1607 s
n: 100000000
nsteps: 100
Elapsed time: 49.1855 s
n: 100000000
nsteps: 100
Elapsed time: 49.1896 s
n: 100000000
nsteps: 100
Elapsed time: 49.2384 s
mpiuser@ip-172-30-8-69:~/cloud$
```