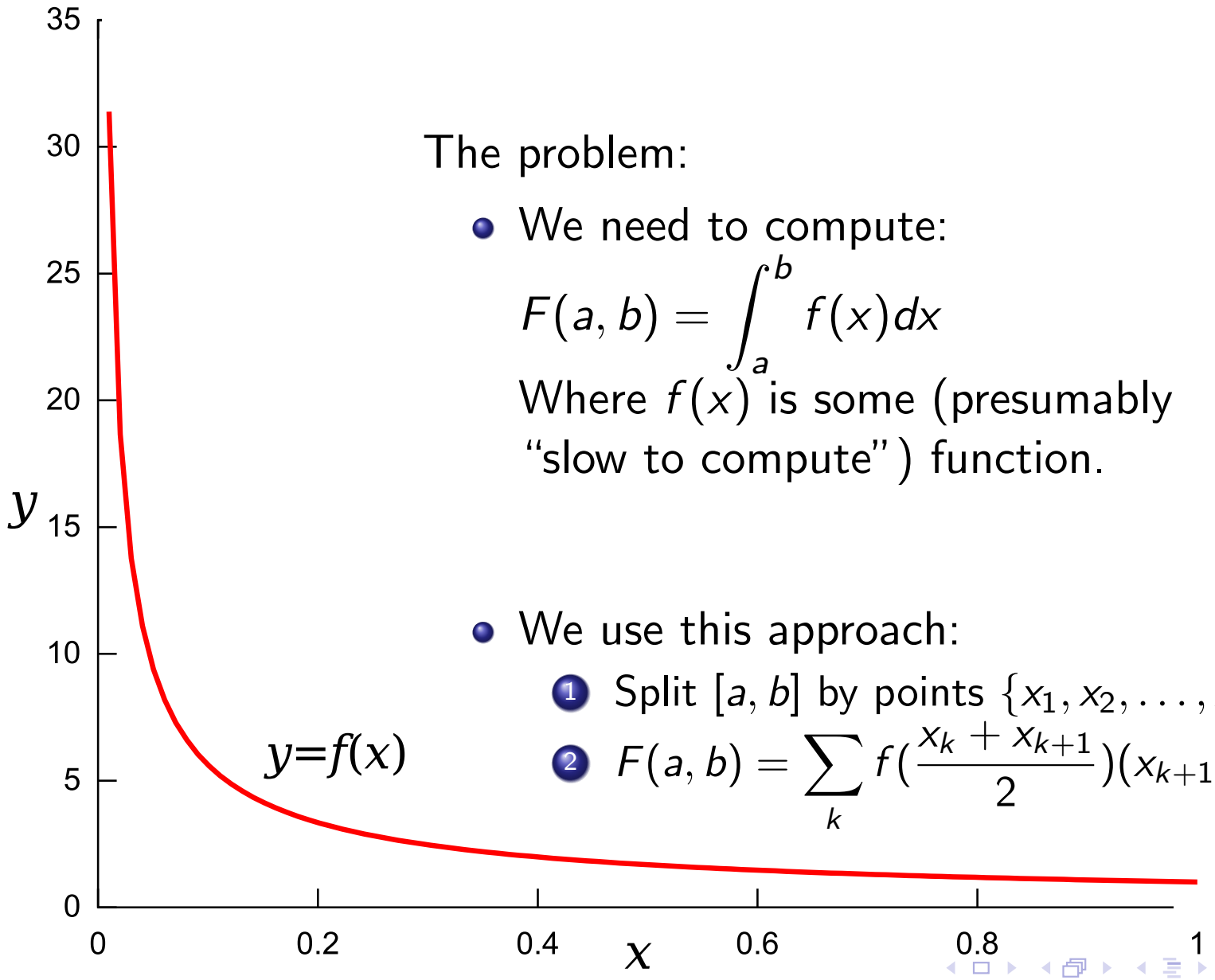
MPI profiling with Allinea MAP

Alexander Gaenko

February 5, 2016

Problem: Calculation of a definite integral.



The problem:

- We need to compute:

  $$F(a,b) = \int_a^b f(x)dx$$

  Where $f(x)$ is some (presumably "slow to compute") function.

- We use this approach:
  1. Split $[a,b]$ by points $\{x_1, x_2, \ldots, x_k\}$
  2. $F(a,b) = \sum_k f(\frac{x_k + x_{k+1}}{2})(x_{k+1} - x_k)$

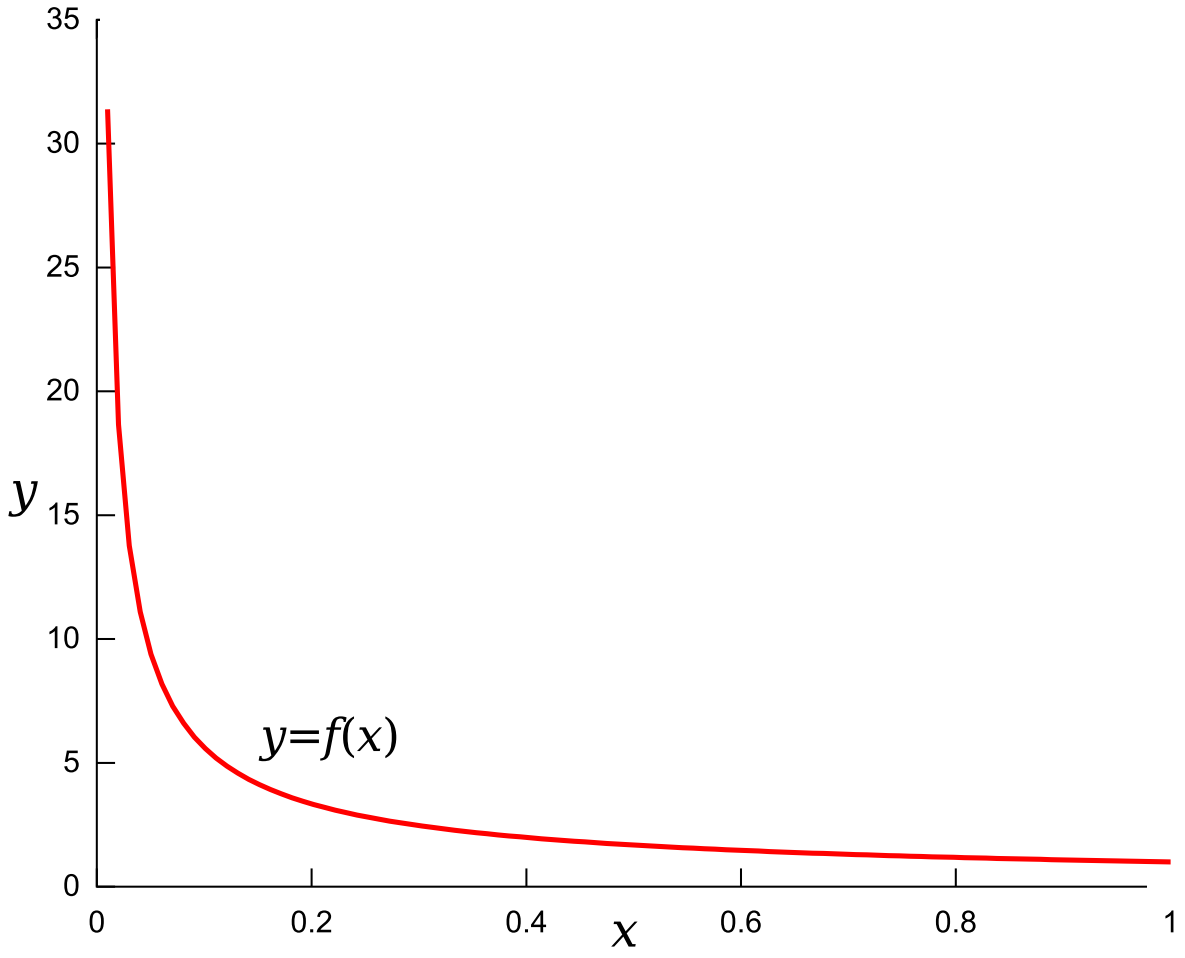
Problem size behavior: how to measure.

Do we even need to parallelize? How fast does the execution time grow with the size of the problem?

```
$ gcc -O3 -o integral_seq.x integral_seq.cxx -L./mylib -lmymath
$ time -a -p ./integral_seq.x 10000
LOREM IPSUM LOREM IPSUM LOREM IPSUM
LOREM IPSUM LOREM IPSUM LOREM IPSUM
LOREM IPSUM LOREM IPSUM LOREM IPSUM
```

Problem size behavior. Graph.

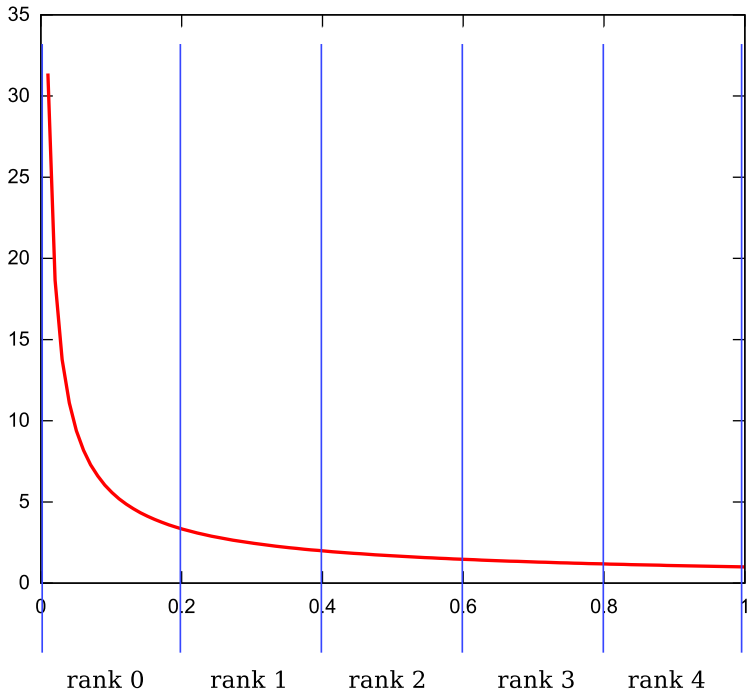
THIS GRAPH IS WRONG



Parallellization: domain decomposition.

Approach:

- Split $[a, b]$ into several domains;
- Compute integrals independently.



$$F(a, b) = \int_a^b f(x)dx$$

1. Assign a process to each domain $[x_k, x_{k+1}]$
2. Let each process compute $F(x_k, x_{k+1})$
3. $F(a, b) = \sum_k F(x_k, x_{k+1})$

?? CODE ??

Parallel performance: how to measure.

Now let's see how much we achieved...

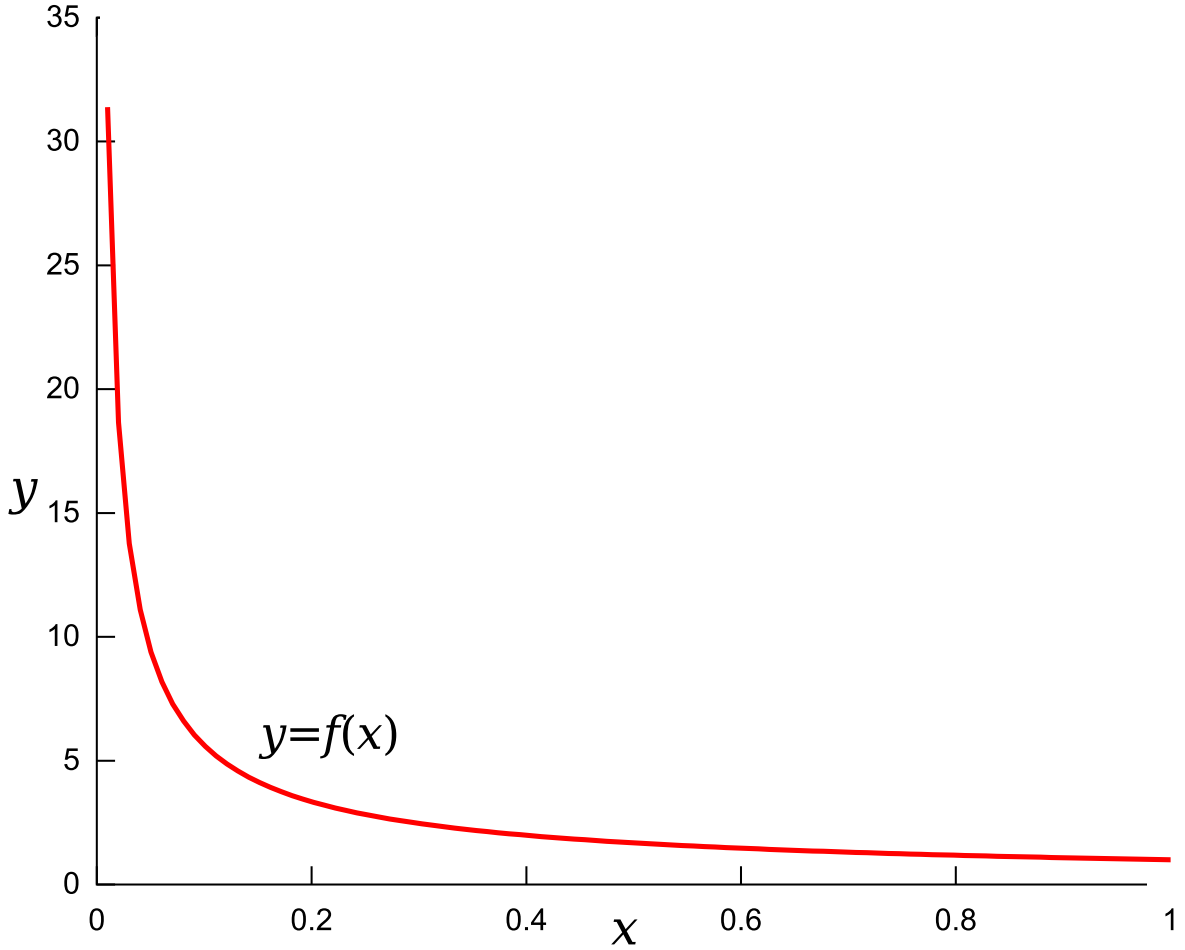
- Strong scaling: as we add processes, how do we fare?
- Weak scaling: as we add *both* processes and work?

```
1 $ mpicc -O3 -o integral_par.x integral_par.cxx -L ./mylib -lmyma
2 $ time -a -p mpirun -np 2 ./integral_seq.x 10000
3 LOREM IPSUM LOREM IPSUM LOREM IPSUM
4 LOREM IPSUM LOREM IPSUM LOREM IPSUM
5 LOREM IPSUM LOREM IPSUM LOREM IPSUM
```

Parallel performance: results

Is there a performance problem?

THIS GRAPH IS WRONG



How does performance analysis work?

How to collect data?

- Instrumentation:
  - Insert timers & counters in the code
  - Requires source or binary processing
- Sampling:
  - Interrupt & check the program at regular intervals
  - Introduces statistical error

What kind of data?

- Profile:
  - Summary information only
  - Relatively small file
- Trace:
  - Detailed recording during the run
  - Potentially huge file
  - Profile can be restored

Allinea MAP does tracing by sampling.

Prepare for profiling

To prepare for profiling, one needs:

- Compile with full optimization
- Generate debugging symbols
- Link with system libs dynamically
  - Usually the default
  - Notable exception: Cray
- On Flux: load `ddt` module

```
1 $ mpicc -g -O3 -o integral_par.x \
2         integral_par.cxx -L ./mylib -lmymath
3 $ module add ddt
```

Running Map: simple way (demo)

1. Get interactive access to a compute node
2. Change to your working directory
3. Optionally, set *sampling interval*
4. Run as you would, prefixed by `map`

```
$ qsub -V -I -X -q flux -l qos=flux,nproc=12 \
       -l walltime=10:0:0 -A account_flux
$ cd $PBS_O_WORKDIR
$ export ALLINEA_SAMPLER_INTERVAL=5
$ map mpirun -np 12 ./integral_par.x 10000
```

Caution:

- Too small interval: large overhead!
- Too large interval: not enough samples!

Running Map: other options

What if you can not or would not run GUI?

- Have slow or non-existing X connection to compute nodes
- Do not want to run interactively

Use `-profile` option.

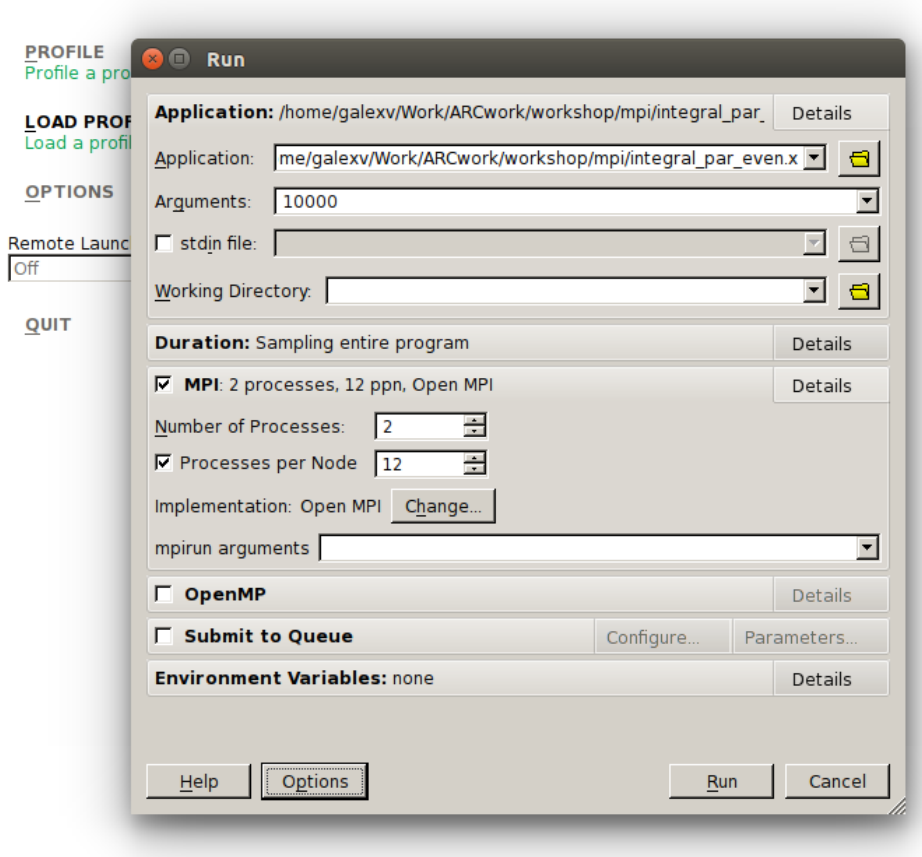
```
1 #PBS -V
2 #PBS -q flux -l qos=flux -A account_flux
3 #PBS -l nproc=12,walltime=10:0:0
4 cd $PBS_O_WORKDIR
5 export ALLINEA_SAMPLER_INTERVAL=5
6 map -profile mpirun -np 12 ./integral_par.x 10000
```

This will create a `*.map` file. Then run from the login node:

```
1 $ map integral_par_even_12p_*.map
```

If you are submitting to a Flux queue...

1. Run `map` from the login node:
   ```
   $ map mpirun -np 12 \
     ./integral_par.x 10000
   ```
2. Set number of processes
3. Check **Submit to queue**
4. Click **Configure...**
5. Load a proper *submission template file* (see next page)
6. Click **OK**
7. Click **Run**



Submission template for Flux

```
1  #PBS -V
2  #PBS -l walltime=WALL_CLOCK_LIMIT_TAG
3  #PBS -l nodes=NUM_NODES_TAG:ppn=PROCS_PER_NODE_TAG
4  #PBS -q QUEUE_TAG -l qos=flux -A account_flux
5  #PBS -o PROGRAM_TAG-allinea.stdout
6  #PBS -e PROGRAM_TAG-allinea.stderr
7
8  cd $PBS_O_WORKDIR
9  AUTO_LAUNCH_TAG
```

Time for a live demo!

SCREENSHOT HERE

- Most of the time is spent in `MPI`
- As the run progresses, *even more* time is spent in `MPI`
- Problem: some processes spend more time in calculating the integrand $f(x)$!
- It's called "Load Imbalance"

Possible solutions:

- Distribute work unevenly (but how?)
- Implement dynamic load balancing

Dynamic load balancing

Manager-Workers approach:

Manager

1. Listen to all workers
2. Worker sent `READY` ?
   send `GO` with a job chunk
3. Worker sent `DONE`?
   add result to the sum
4. No more job chunks?
   send `STOP` to the worker
5. No more workers?
   we are done

Worker

1. Send `READY` to the Manager
2. Listen to the Manager
3. Manager sent `GO` ?
   - Get job chunk
   - Do the calculation
   - Send `DONE` with result to the Manager
   - Go to (1)
4. Manager sent `STOP`?
   exit.

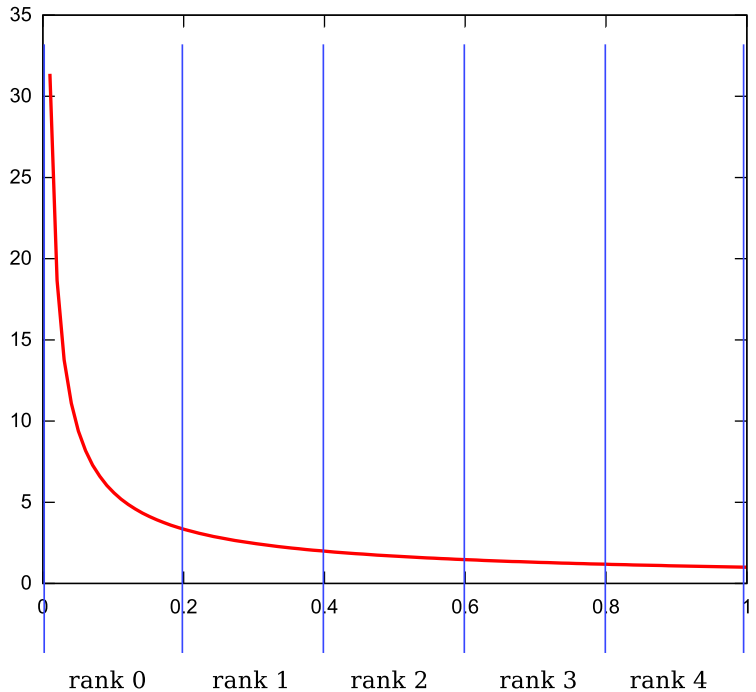
Strong scaling graph.

Map demo and screenshot.

The problem and the first approach

$$I(a,b) = \int_a^b x^{-3/4} dx$$

Analytic solution to cross-check: $I(a,b) = 4(b^{1/4} - a^{1/4})$

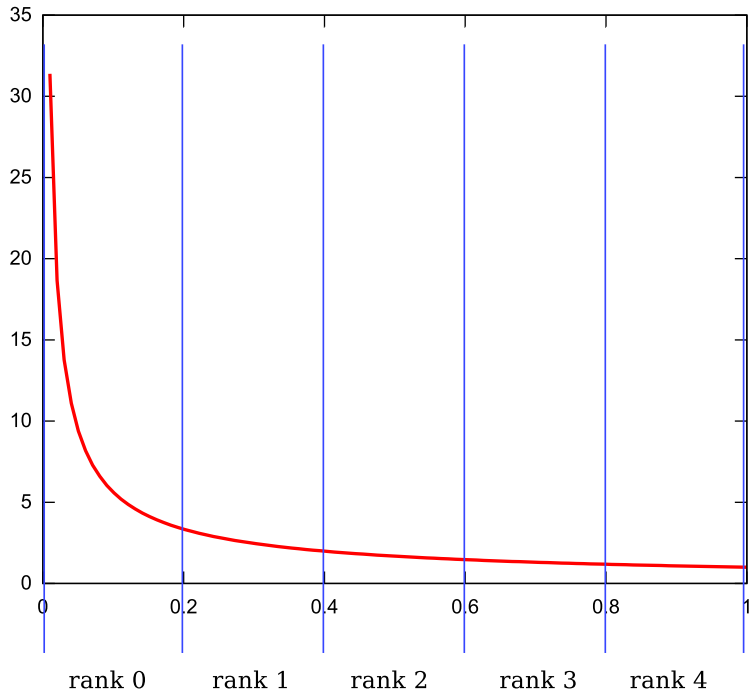


- Split $[a,b]$ equally between workers
- $I(a,b) = \sum_k I(x_k, x_{k+1})$

The problem and the first approach

$$I(a,b) = \int_a^b x^{-3/4} dx$$

Analytic solution to cross-check: $I(a,b) = 4(b^{1/4} - a^{1/4})$



- Split $[a,b]$ equally between workers
- $I(a,b) = \sum_k I(x_k, x_{k+1})$
- Caveat: integrand takes more time to compute at low x!

Dynamic load balancing

Try various chunk sizes, various number of processes.
Where are the bottlenecks?

(Alternative: give smaller chunks to low X. Scalability. Profile.)