Jure Pečar

Compute Cluster Intermediate Workshop

slurm

workload manager

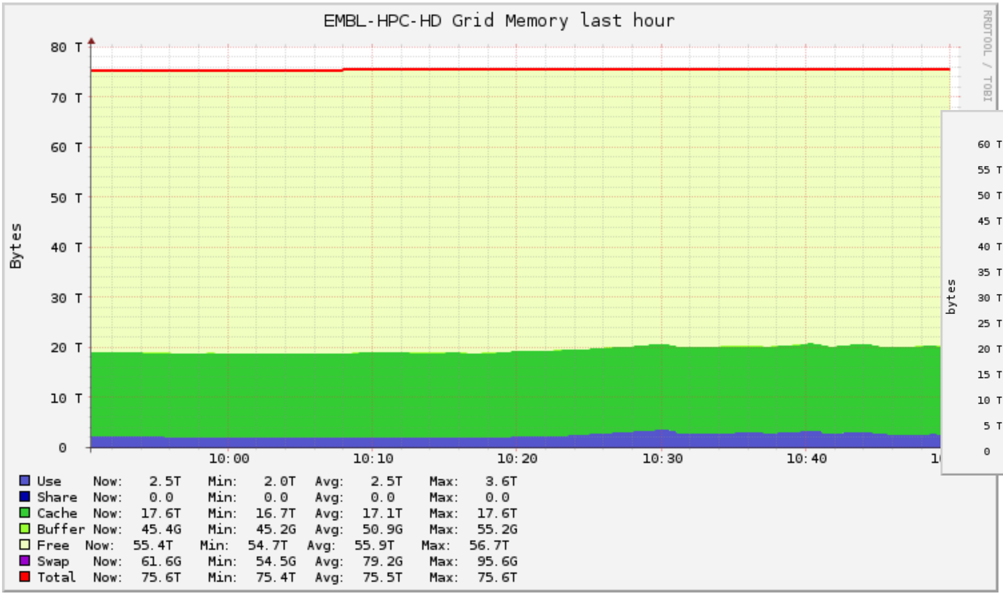
Recap

Recap

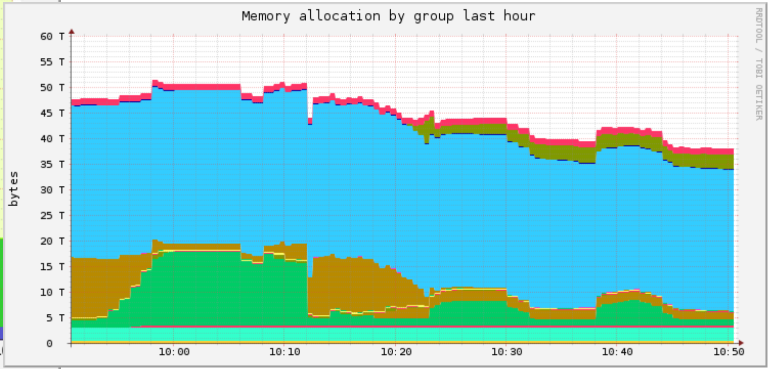
- Cluster is a shared resource
- Tune your resource requests to maximize utilization
    - sacct, seff, or --mail
- Load modues within job script
- Always try to move data as close to compute as possible
    - Copy your data to $TMPDIR as first step in your job
    - Copy your results from $TMPDIR as last step

```bash
#!/bin/bash
#SBATCH --time=00-00:06:00
#SBATCH --mem=4000M
#SBATCH --nodes=1
#SBATCH --tmp=1G
#SBATCH --gres=tmp:1G
#SBATCH --output=bwa.out
#SBATCH --open-mode=append

## load required modules
module load SAMtools BWA

## copy data to /tmp and change directory to /tmp
cp /g/huber/users/msmith/embl_hpc/Ecoli_genome.fa.gz $TMPDIR
cp /g/huber/users/msmith/embl_hpc/reads_*.fq.gz $TMPDIR
cd $TMPDIR

## create an index
bwa index -p ecoli Ecoli_genome.fa.gz

## perform alignment
bwa mem ecoli reads_1.fq.gz reads_2.fq.gz > aligned.sam

## create a compressed BAM file
samtools view -b aligned.sam > aligned.bam

## copy results back to where job was subbmited from
cp aligned.bam $SLURM_SUBMIT_DIR/
```

On topic of memory …



```
🟦 Use    Now:   2.5T   Min:   2.0T   Avg:   2.5T   Max:   3.6T
🟦 Share  Now:   0.0    Min:   0.0    Avg:   0.0    Max:   0.0
🟩 Cache  Now:  17.6T   Min:  16.7T   Avg:  17.1T   Max:  17.6T
🟩 Buffer Now:  45.4G   Min:  45.2G   Avg:  50.9G   Max:  55.2G
🟨 Free  Now:  55.4T   Min:  54.7T   Avg:  55.9T   Max:  56.7T
🟪 Swap   Now:  61.6G   Min:  54.5G   Avg:  79.2G   Max:  95.6G
🟥 Total  Now:  75.6T   Min:  75.4T   Avg:  75.5T   Max:  75.6T
```



Types of Memory

- RES (or RSS) – resident (set) size
  - Indicates used physical memory
- SHR – shared memory
  - Memory that can be shared with other processes, like libraries
- VIRT – virtual size
  - res + shr + memory mapped files
  - Indicates how much memory process is able to access

← this goes in sbatch --mem

Goal:

Minimize time to result

Challenge:

Understanding bottlenecks

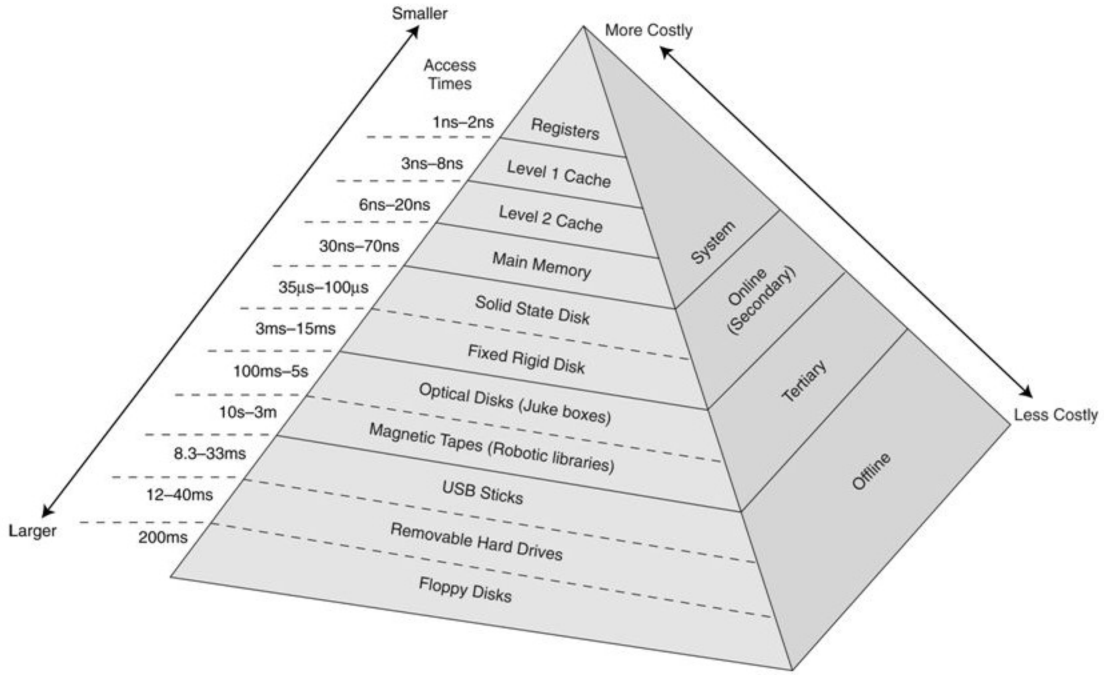
&

Identifying and exploiting parallelism

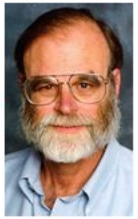
IO bottlenecks

In Lifesciences bottleneck is the IO

- Data formats are designed without regard for the underlying storage system
- What works on your laptop might not work well on cluster
- What exactly is “bad IO”?
- IO is characterized by bandwidth and latency


Smaller
Access Times
Larger
More Costly
Less Costly
1ns–2ns
3ns–8ns
6ns–20ns
30ns–70ns
35µs–100µs
3ms–15ms
100ms–5s
10s–3m
8.3–33ms
12–40ms
200ms
Registers
Level 1 Cache
Level 2 Cache
Main Memory
Solid State Disk
Fixed Rigid Disk
Optical Disks (Juke boxes)
Magnetic Tapes (Robotic libraries)
USB Sticks
Removable Hard Drives
Floppy Disks
System
Online (Secondary)
Tertiary
Offline

Jim Gray's Storage Latency Analogy: How Far Away is the Data?



10**9 tape	Andromeda		2,000yr
10**6 disk	Pluto		2yr
100 Memory	Pittsburgh		1.5h
10 On board cache	This building		10min
2 on chip cache	This room		1min
1 registers	In my head		

In Lifesciences bottleneck is the IO

- Network attached storage: ok on bandwidth, poor on latency

```
pecar@login:~$ ping -c 5 fhgfs1
PING fhgfs1.cluster.embl.de (10.11.12.87) 56(84) bytes of data.
64 bytes from fhgfs1.cluster.embl.de (10.11.12.87): icmp_seq=1 ttl=64 time=0.106 ms
64 bytes from fhgfs1.cluster.embl.de (10.11.12.87): icmp_seq=2 ttl=64 time=0.118 ms
64 bytes from fhgfs1.cluster.embl.de (10.11.12.87): icmp_seq=3 ttl=64 time=0.284 ms
64 bytes from fhgfs1.cluster.embl.de (10.11.12.87): icmp_seq=4 ttl=64 time=0.089 ms
64 bytes from fhgfs1.cluster.embl.de (10.11.12.87): icmp_seq=5 ttl=64 time=0.097 ms

--- fhgfs1.cluster.embl.de ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3999ms
rtt min/avg/max/mdev = 0.089/0.138/0.284/0.074 ms
```

IO operations per second

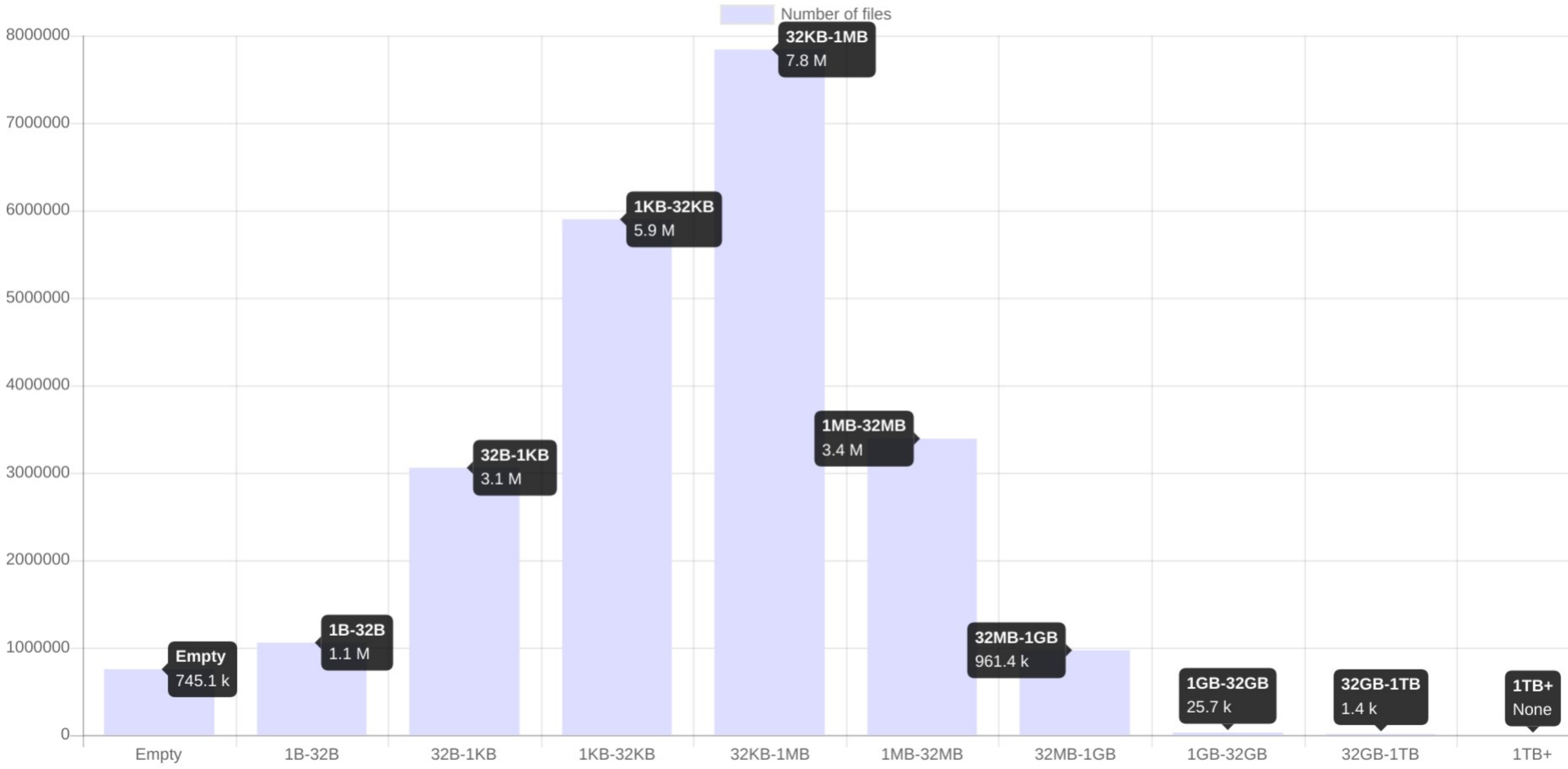
- 0.138ms rtt→ 1000/0.138 = 7246 iops
- We have 10Gb link on login node, so:
- 10240/7246 = 1.41Mb or 180KB per IO

- If you transfer less than 180KB per IO, you cannot make the full use of bandwidth

Metadata IO

- Cannot be big, limited by network rtt
- Is typically synchronous on the storage side → limited by storage latencies

- Any massive metadata IO sooner or later results in IOwait, either on client or on server



Number of files
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
Empty
745.1 k
1B-32B
1.1 M
32B-1KB
3.1 M
1KB-32KB
5.9 M
32KB-1MB
7.8 M
1MB-32MB
3.4 M
32MB-1GB
961.4 k
1GB-32GB
25.7 k
32GB-1TB
1.4 k
1TB+
None
Empty
1B-32B
32B-1KB
1KB-32KB
32KB-1MB
1MB-32MB
32MB-1GB
1GB-32GB
32GB-1TB
1TB+

Avoid network latency

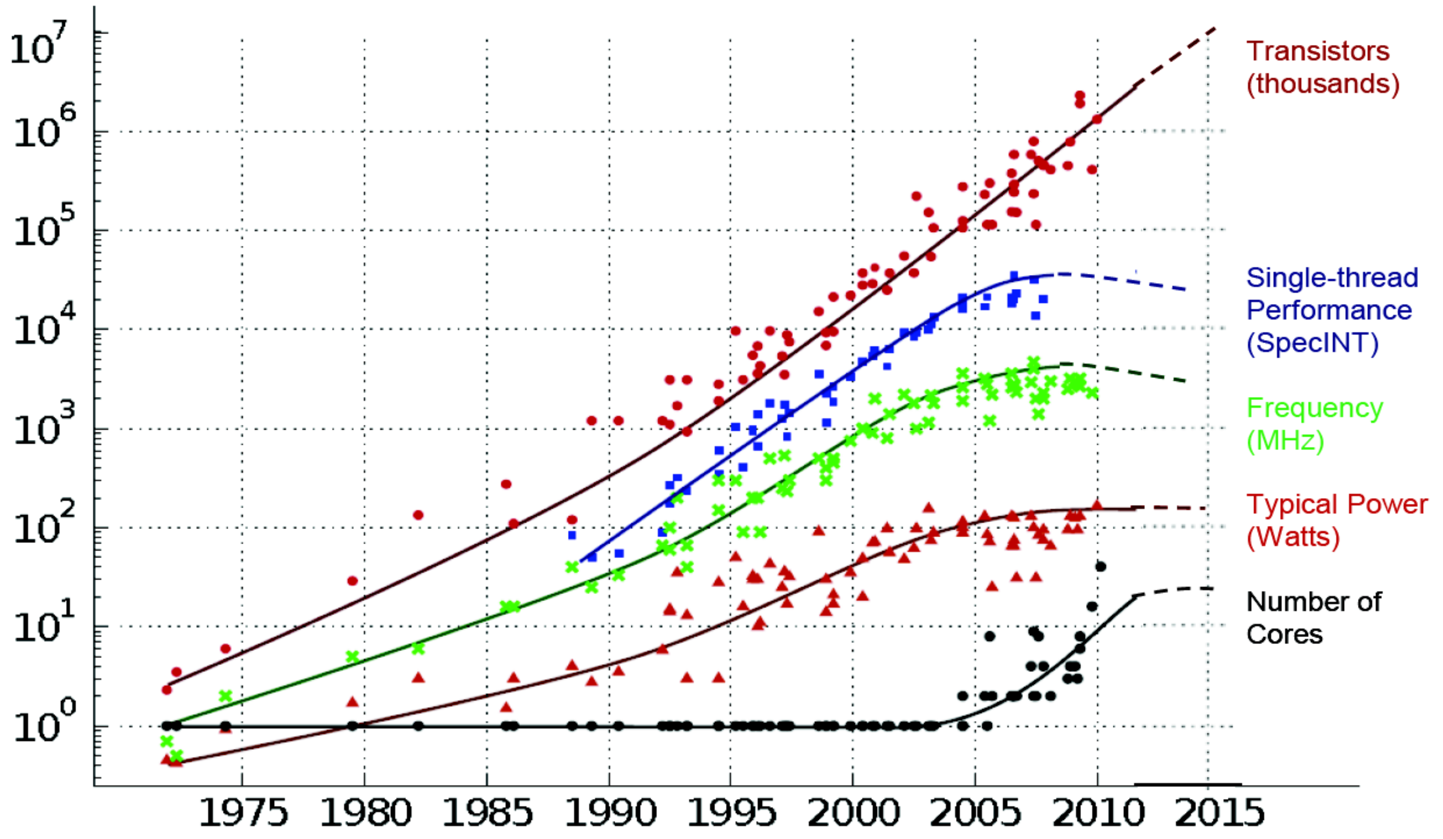
By copying your working data to $TMPDIR

IO takeout

- Network attached storage – good for bulk transfers
- Local storage – good for small io
  - Especially flash (ssd, nvme)

Parallelism

35 YEARS OF MICROPROCESSOR TREND DATA



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Types of parallelism

- Bit level
- Instruction level
- Task level
- Data level

Bit level parallelism

Bit level parallelism - SIMD

- Packing more data into single instruction
  - So called “vector extensions”
  - AVX (128bit)
  - AVX2 (256bit)
  - AVX512
- Best applied when writing code
- Compilers attempt to do their best at autovectorisation

Node Features

- Instruction set: avx, avx2, avx512
- CPU family: nehalem, sandybridge, haswell, broadwell, skylake, epyc
- CPU frequency: cpuX.XGHz
- Hyperthreading: HT or noHT
- By GPU type: gpu=1080Ti or gpu=P100

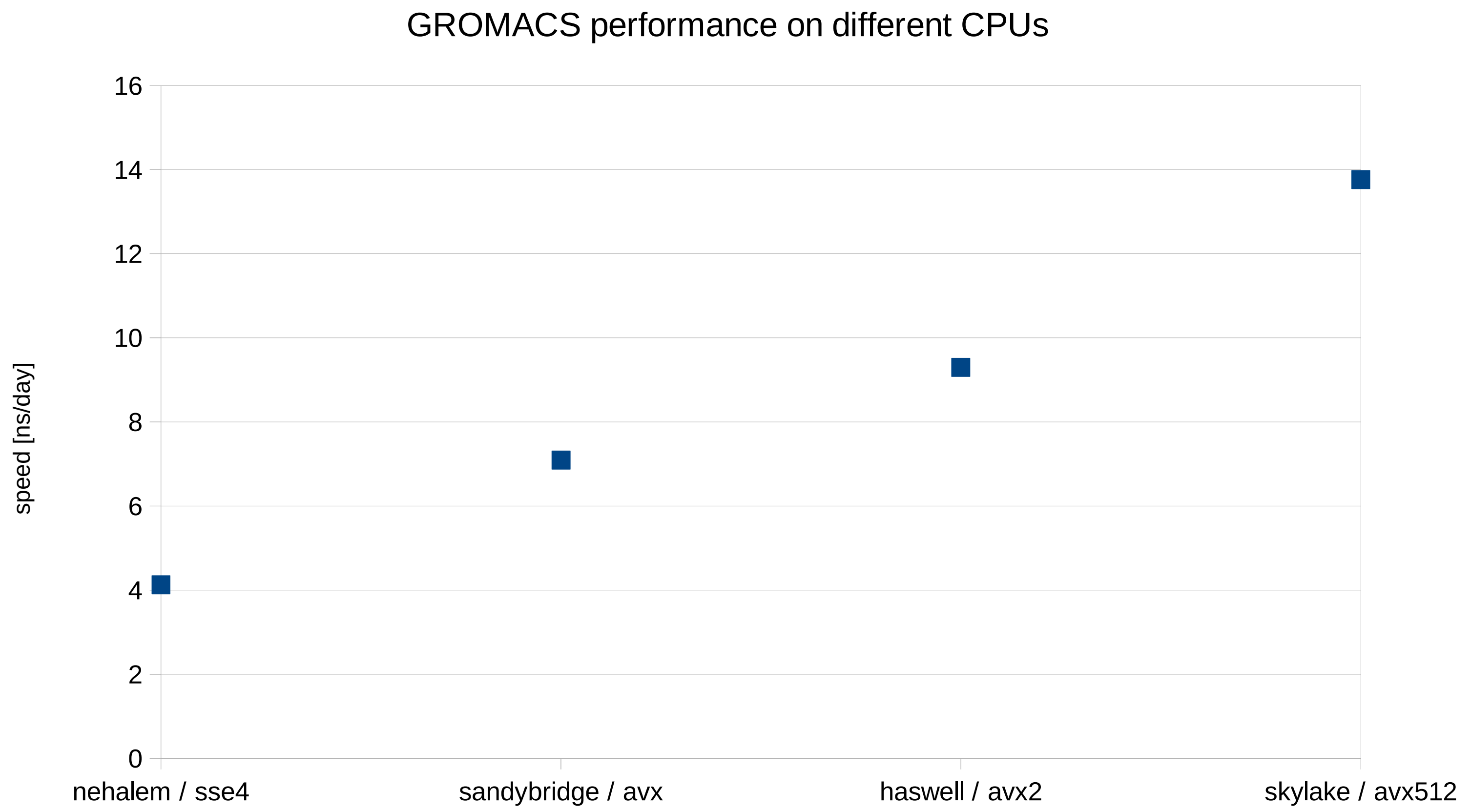
Examples

```
git clone https://git.embl.de/grp-bio-it/embl_hpc -b march2019
```

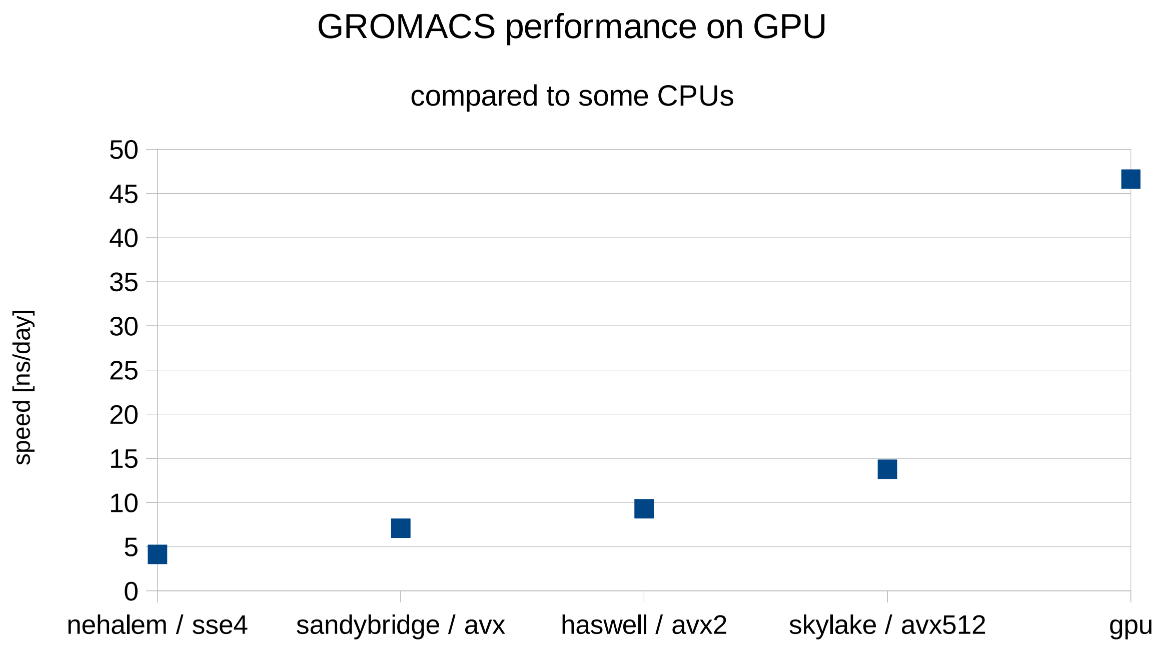
Example: GROMACS

- Popular molecular dynamics software
- Example provided by @pchen
  - “protein in water”
- See exercises/gromacs/job.sh

```
#!/bin/bash

#SBATCH -J gromacs
#SBATCH -N 1
#SBATCH -c 8
#SBATCH --hint=nomultithread
#SBATCH -C avx #or avx2 or avx512
#SBATCH -t 04:00


module load GROMACS/2018.1-foss-2017b

cp /g/its/home/pecar/benchmarks/pchen_gromacs/1OVA-AB.tpr $TMPDIR
cd $TMPDIR

gmx mdrun -s 1OVA-AB.tpr -nsteps 5000 -ntmpi 1

tail -5 md.log
```


GROMACS performance on different CPUs
speed [ns/day]
0
2
4
6
8
10
12
14
16
nehalem / sse4
sandybridge / avx
haswell / avx2
skylake / avx512

Hint: these kind of codes tend to work well on GPUs



Instruction level parallelism

Instruction level parallelism

- Mostly a concern of cpu designers and compiler developers
- Symmetric multi-threading (or hyperthreading) is an example

Task level parallelism

Task level parallelism

- Running different tasks on same data

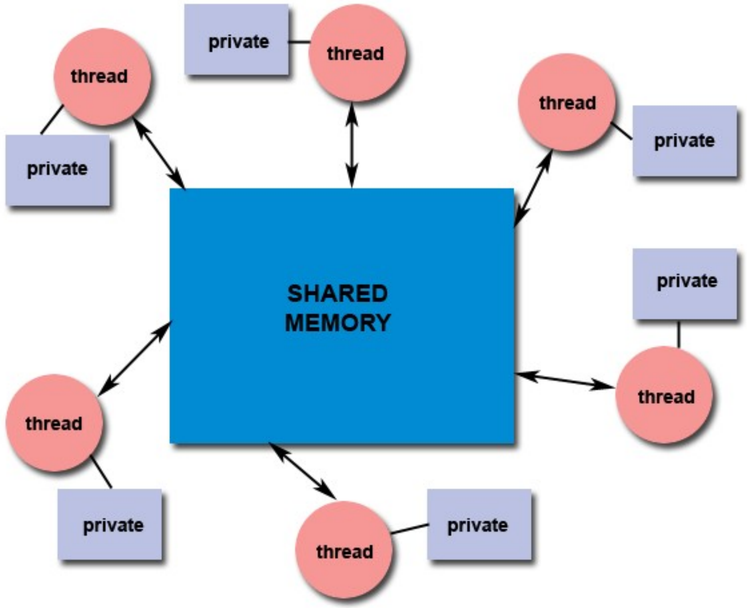
- Many threading libraries and all MPI stacks allow you to do that

Data level parallelism

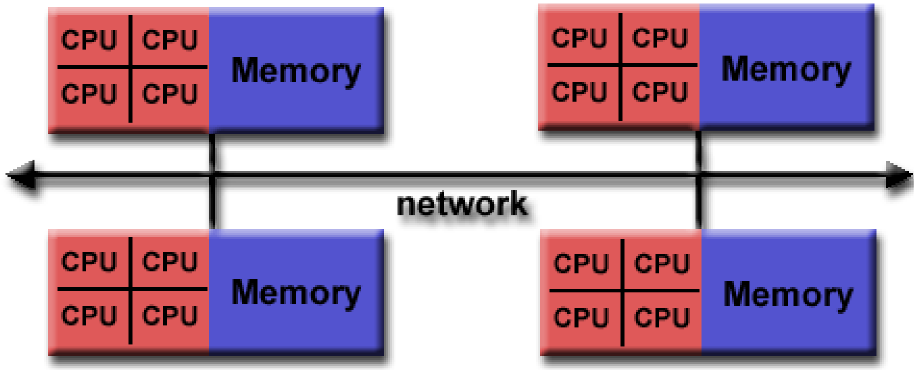
Data level paralelism

- Running same task on independent chunks of data
- On a code level implemented with
  - POSIX threads
  - CUDA
  - OpenMP, OpenACC
  - MPI
  - PGAS
  - ...

- Python:
  - 14 entries on python multiprocessing wiki
- R
  - Nice post on r-bloggers: https://www.r-bloggers.com/a-guide-to-parallelism-in-r/



VS



Posix threads
OpenMP, OpenACC
PGAS

CUDA (across multiple GPUs)
MPI

Finding the best number of threads

- The more the better, right?

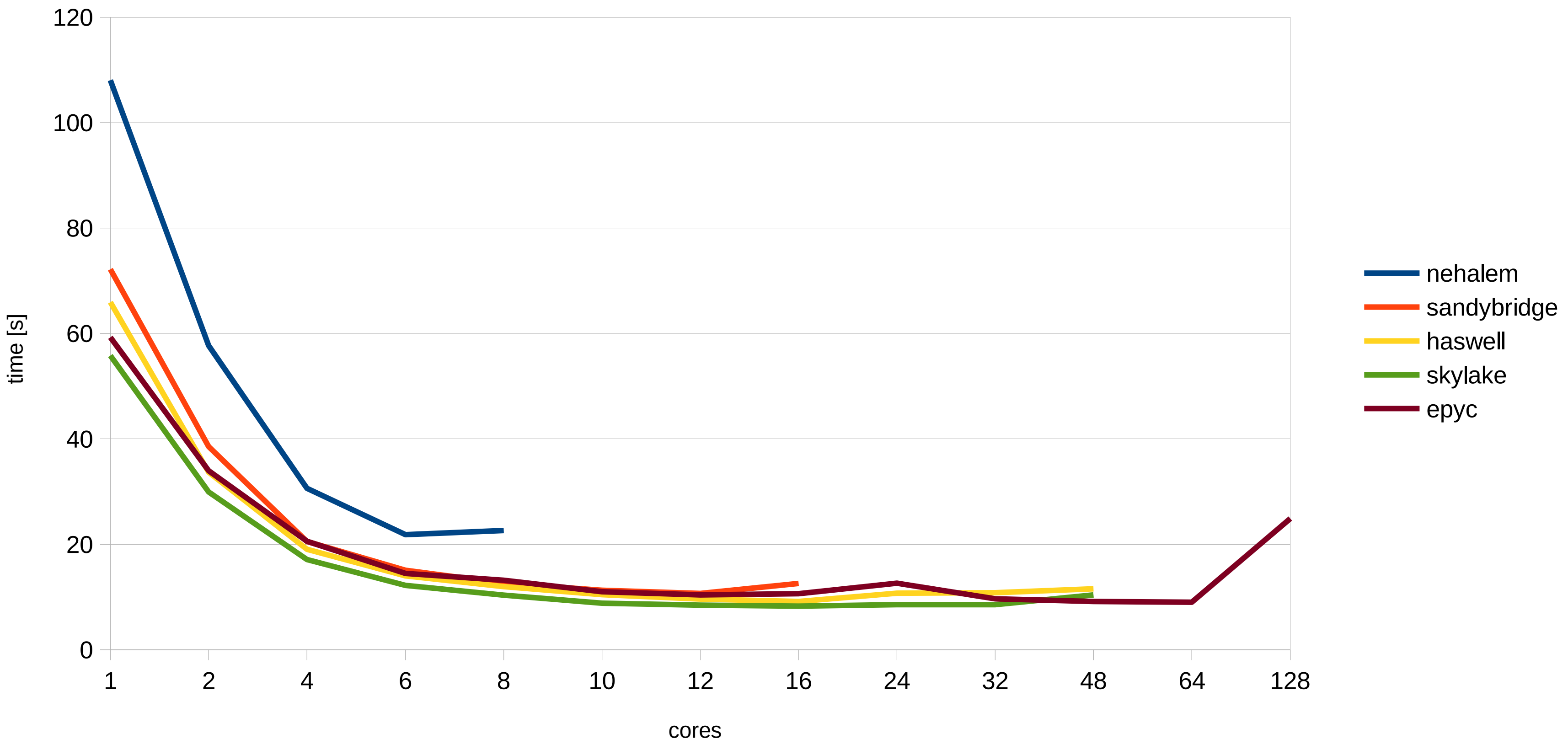
- Well … no.

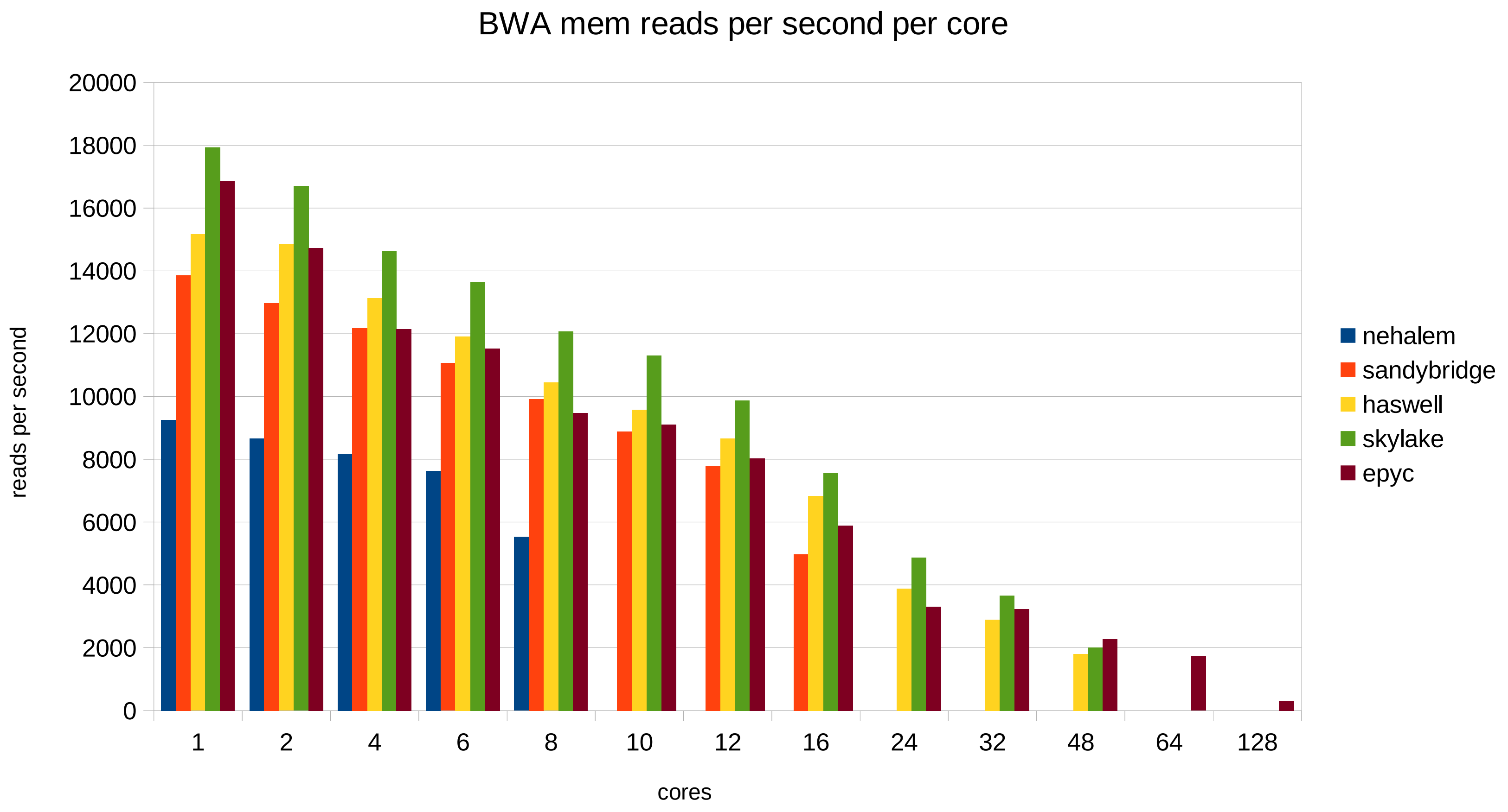
Example: bwa

- Burrows-Wheeler Alignment Tool
  - Bwa supports multithreading
- Example provided by @msmith
  - Based on Ecoli
- See examples/bwa/bwa_batch.sh

```
#!/bin/bash
#SBATCH -J bwa
#SBATCH --time=00-00:06:00
#SBATCH --mem=4000M
#SBATCH --nodes=1
#SBATCH -c 1 #vary this 1..128
#SBATCH --tmp=1G
#SBATCH --gres=tmp:1G
#SBATCH --output=bwa.out
#SBATCH --open-mode=append

module load SAMtools BWA

cp /g/its/home/pecar/benchmarks/msmith_bwa/Ecoli_genome.fa.gz $TMPDIR
cp /g/its/home/pecar/benchmarks/msmith_bwa/reads_*.fq.gz $TMPDIR
cd $TMPDIR

bwa index -p ecoli Ecoli_genome.fa.gz

bwa mem -t $SLURM_CPUS_PER_TASK ecoli reads_1.fq.gz reads_2.fq.gz > aligned.sam

samtools view -b aligned.sam > aligned.bam
```

BWA mem scalability on Ecoli




BWA mem reads per second per core
reads per second
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
1
2
4
6
8
10
12
16
24
32
48
64
128
cores
nehalem
sandybridge
haswell
skylake
epyc

So ...

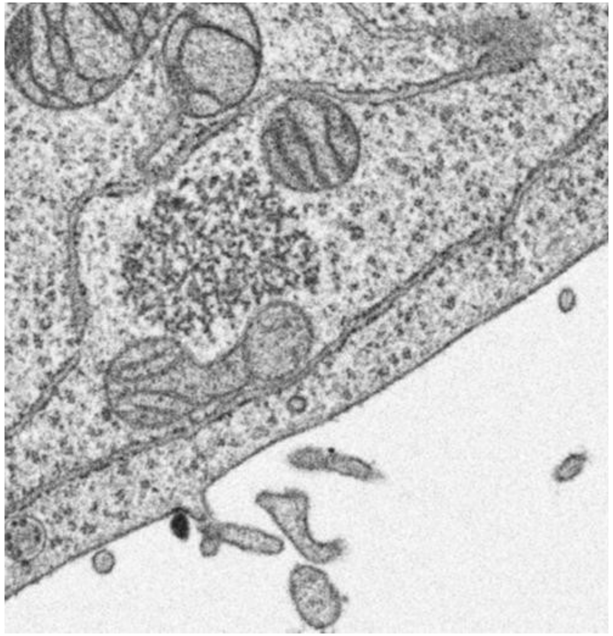
- Best for this case looks like around 4
  - Can vary based on data and algorithms
- Why?
  - Topic for advanced course ;)

Takeout

- Aim for smaller number of thredas
  - Unless your software scales really well
- So you can fit more jobs on the same number of cpus
- And you get better throughput

Example: ImageJ

- Java based image analysis and processing
- Example provided by @tischer
- Sample from Focussed-Ion-Beam Scanning Electron Microscopy (FIB-SEM), 10x10x10 nm
- Random forest classifier trained to distinguish ER from the rest

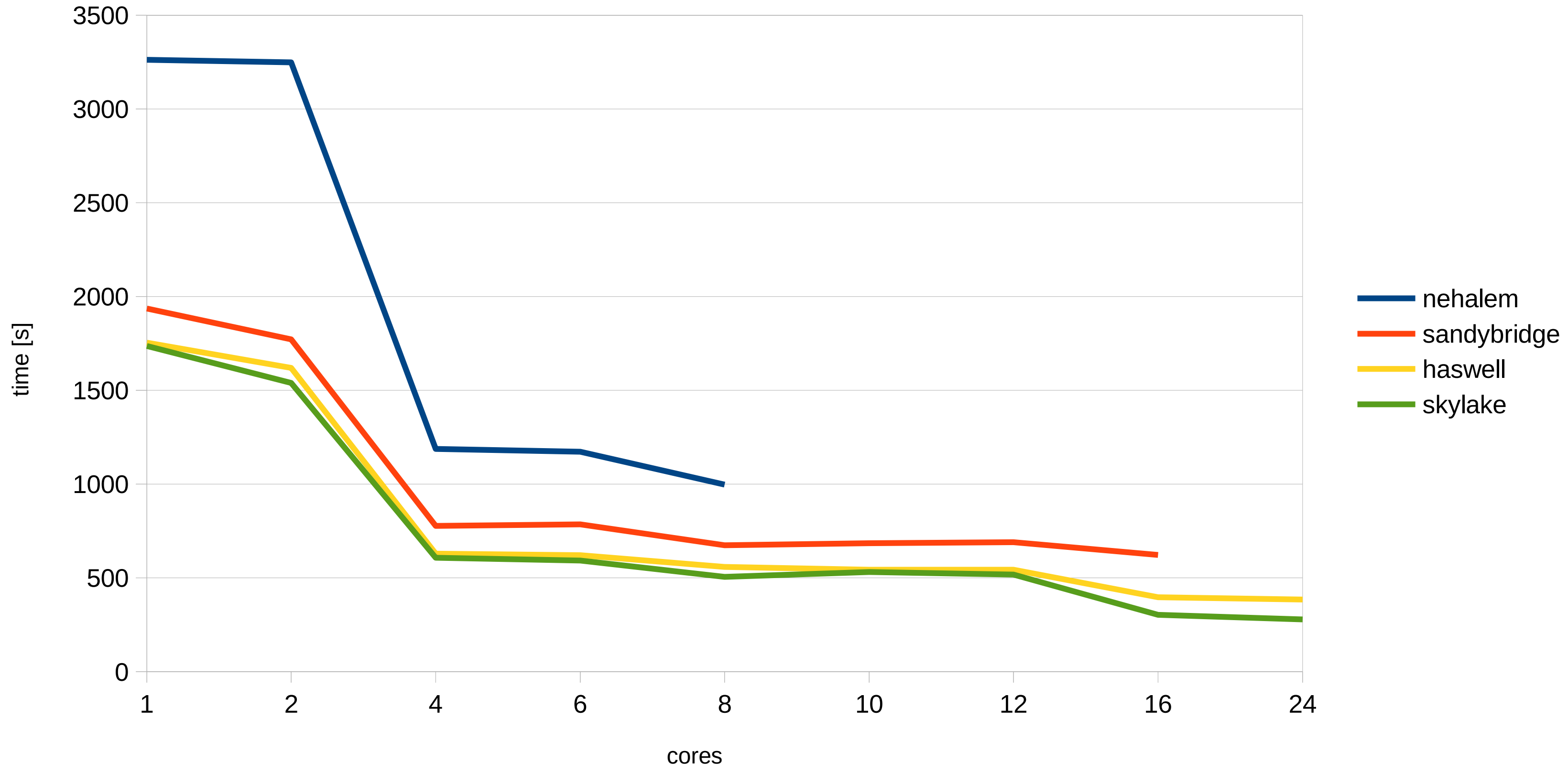


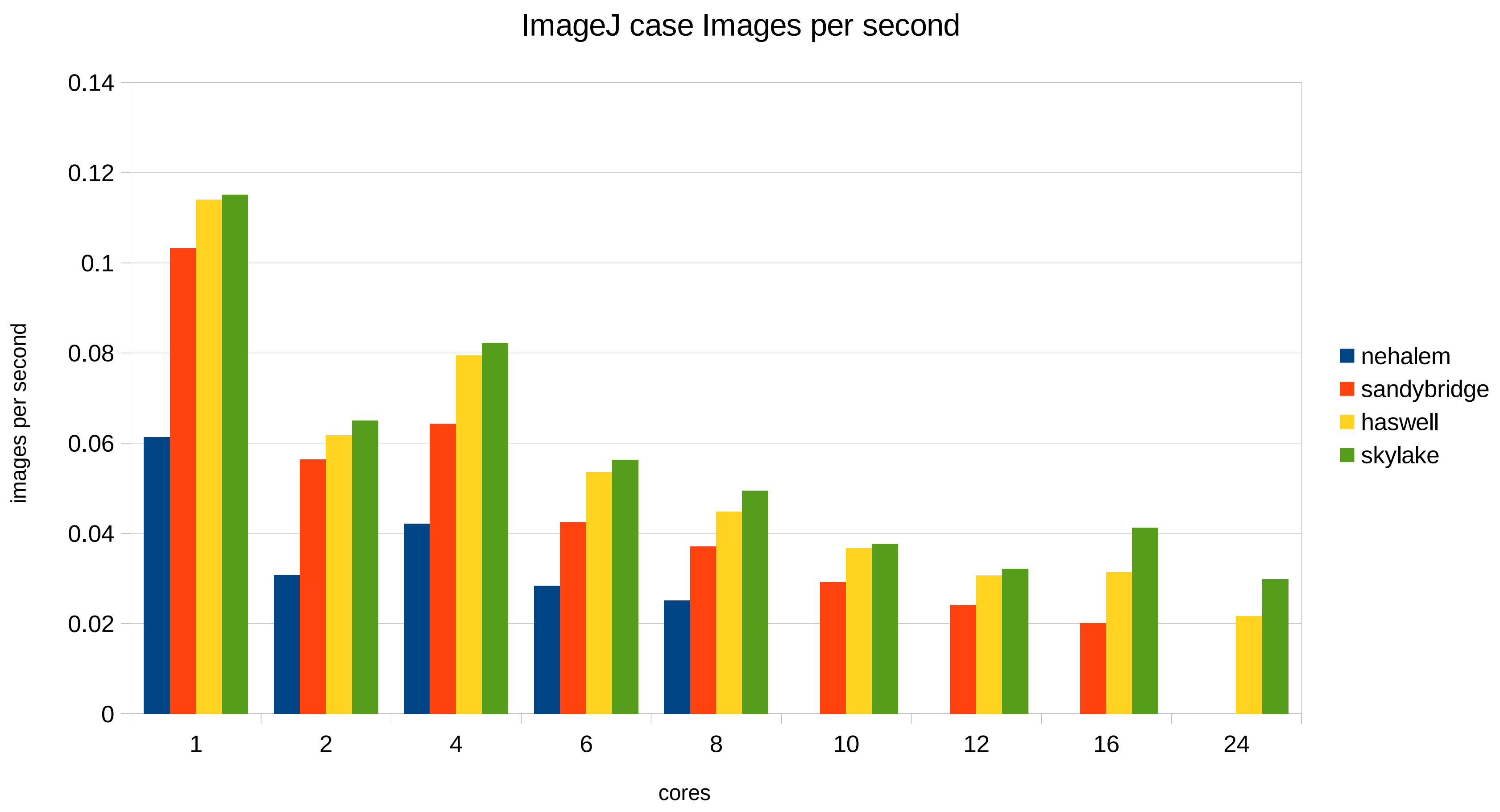
Challenge

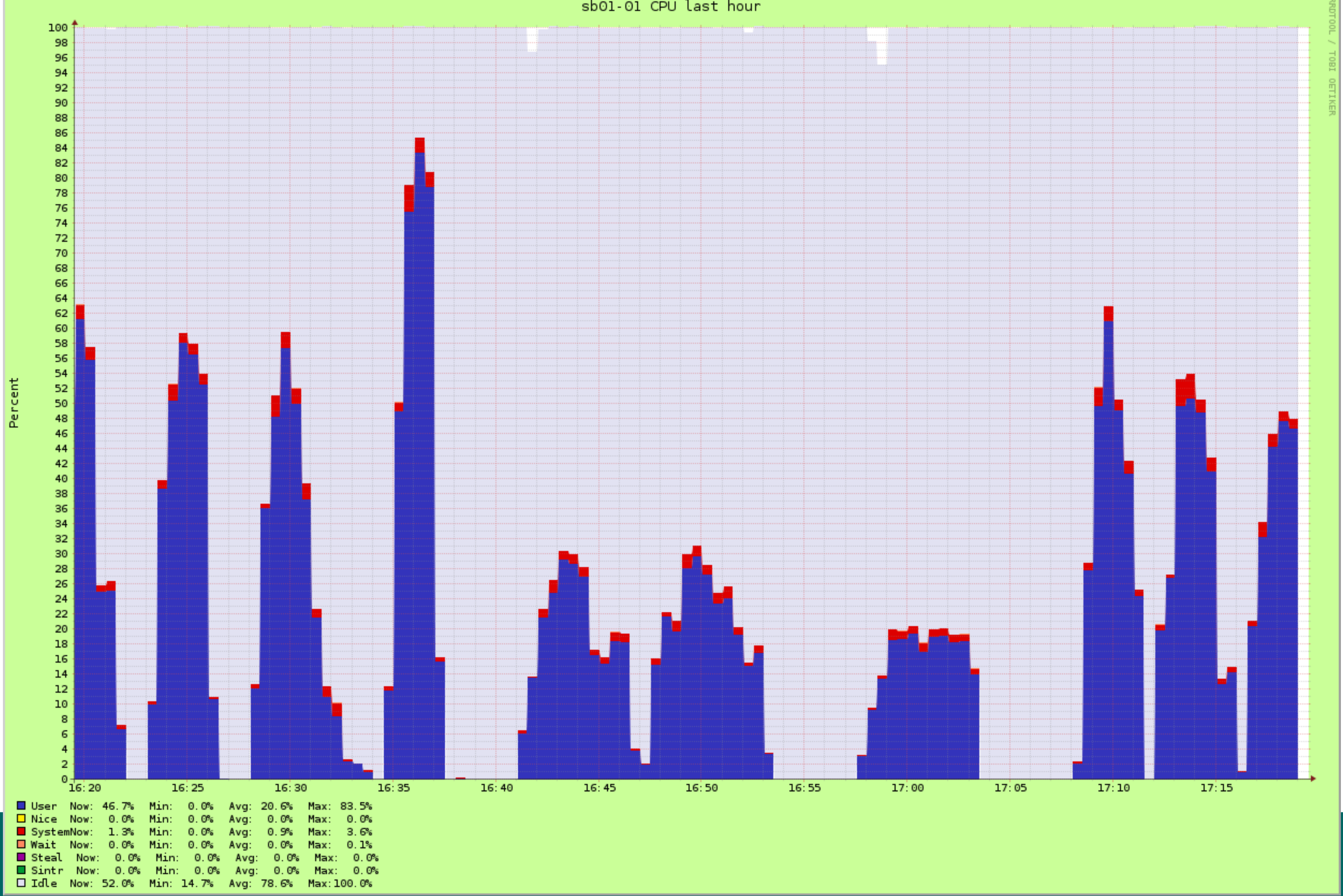
- JVM does its own hardware abstraction and thread management
- Presents whole new dimension in problem space
- See examples/imagej/serial.job

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 24
#SBATCH --mem 180000
#SBATCH -t 0-01:00:00

module load Java
module load X11

mkdir -p ~/.imagej
cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/IJ_Prefs.txt ~/.imagej/

cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/fib-sem--cell--8x8x8nm.tif
$TMPDIR/
cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/bg-er.classifier $TMPDIR/

cd $TMPDIR

START_TIME=$SECONDS

/g/almf/software/Fiji.app/ImageJ-linux64 --mem=32000M --ij2 --allow-multiple --headless --
run  "Apply Classifier" "inputImageFile='fib-sem--cell--
8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-
er.classifier',dataSetID='fib-sem--cell--8x8x8nm',outputModality='Save class probabilities
as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,0,200,0,0',numWo
rkers='24'"
```

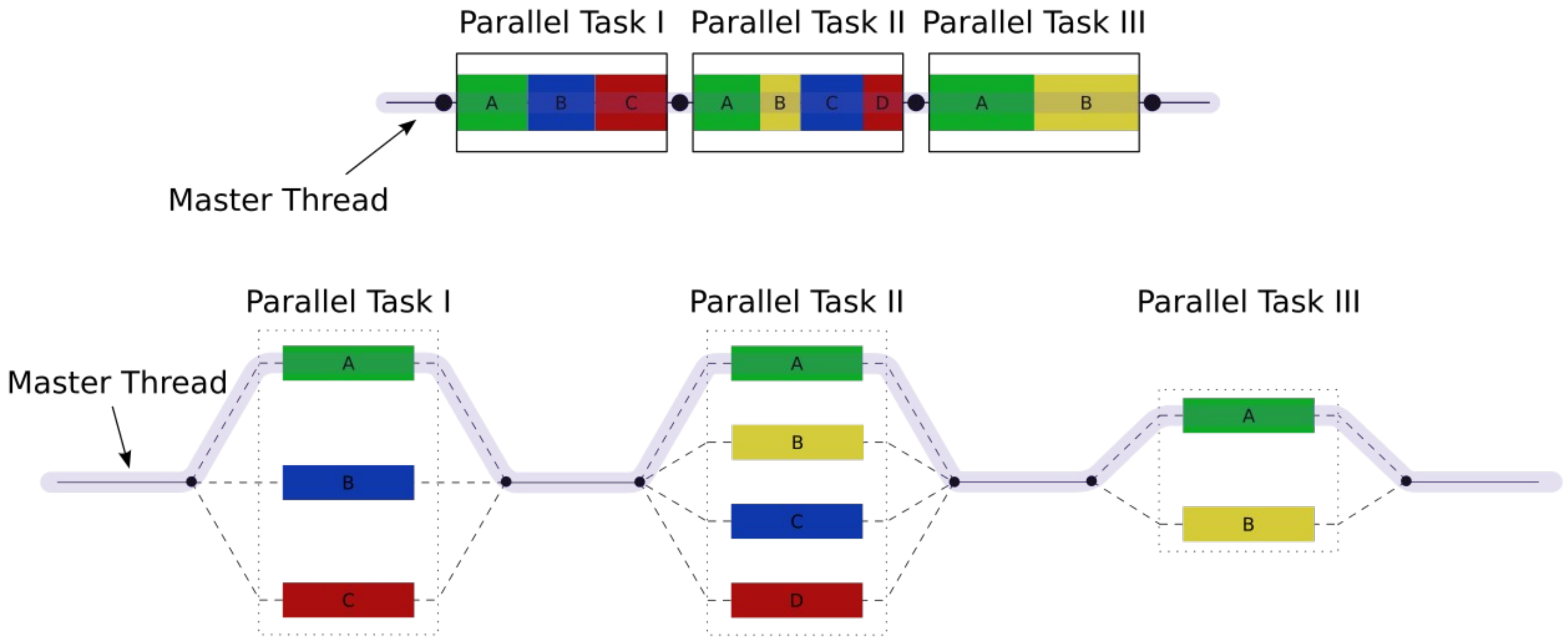
ImageJ case scaling

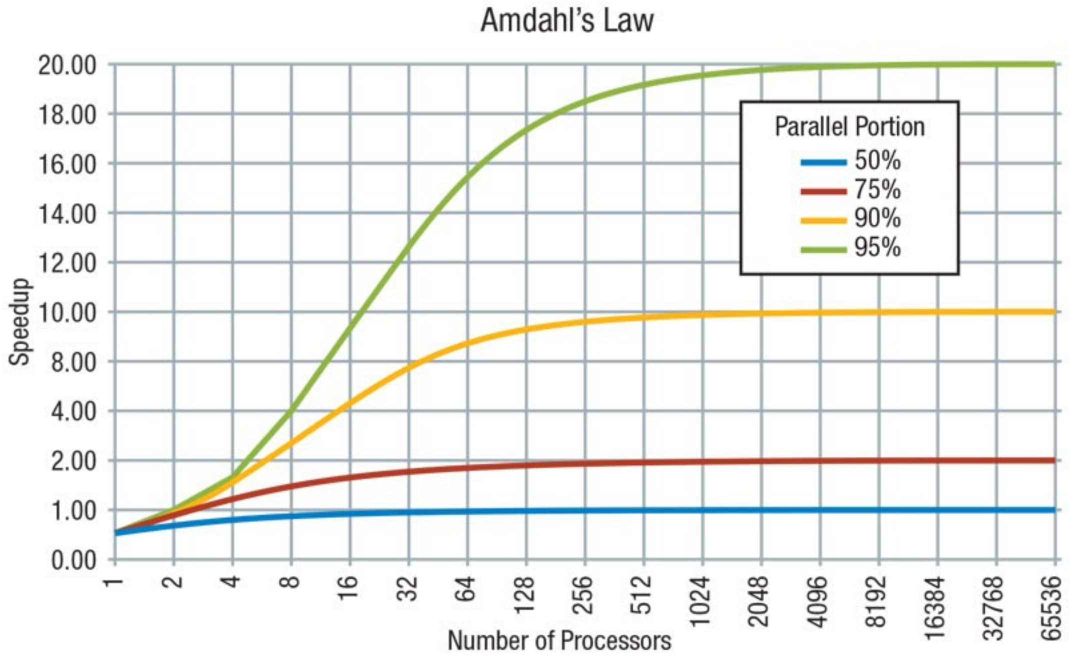



ImageJ case Images per second
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
images per second
1
2
4
6
8
10
12
16
24
cores
nehalem
sandybridge
haswell
skylake



```
■ User  Now: 46.7%  Min:  0.0%  Avg: 20.6%  Max: 83.5%
■ Nice  Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.0%
■ SystemNow:  1.3%  Min:  0.0%  Avg:  0.9%  Max:  3.6%
■ Wait  Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.1%
■ Steal  Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.0%
■ Sintr  Now:  0.0%  Min:  0.0%  Avg:  0.0%  Max:  0.0%
□ Idle  Now: 52.0%  Min: 14.7%  Avg: 78.6%  Max:100.0%
```

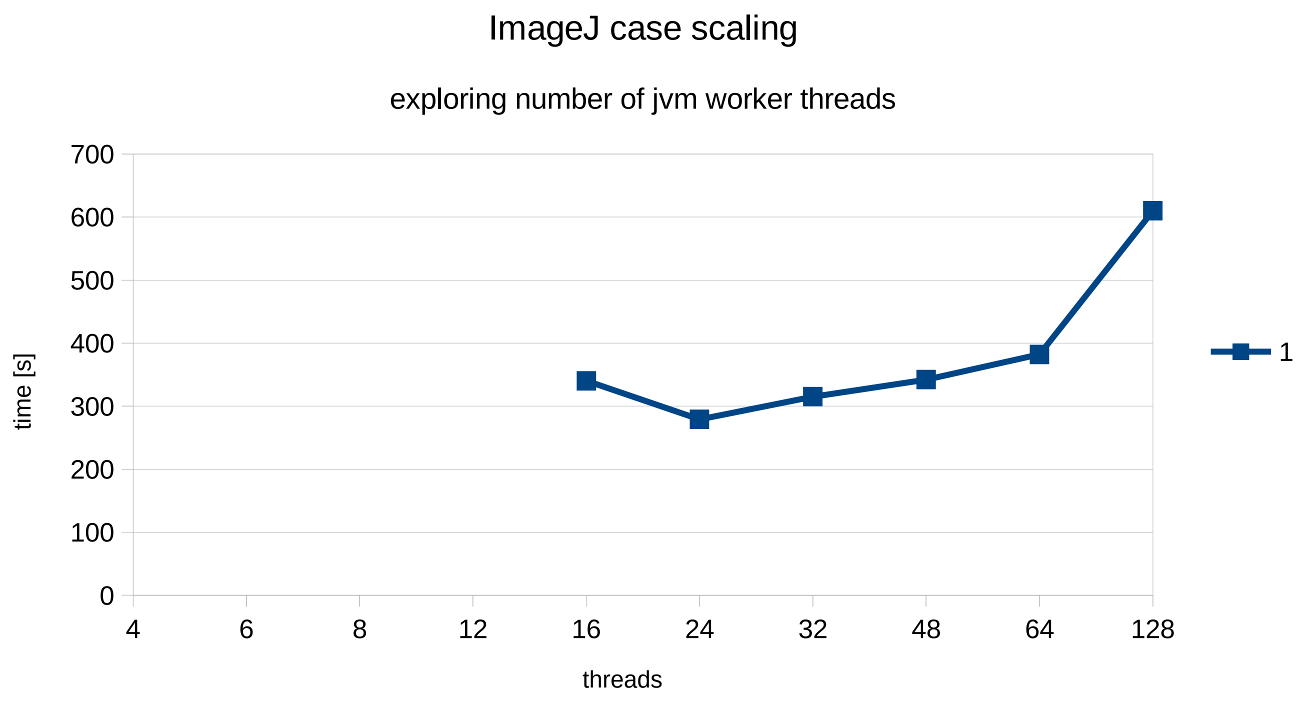

Parallel Task I
Parallel Task II
Parallel Task III
A
B
C
A
B
C
D
A
B
Master Thread
Parallel Task I
Parallel Task II
Parallel Task III
Master Thread
A
B
C
A
B
C
D
A
B


Amdahl's Law
Speedup
20.00
18.00
16.00
14.00
12.00
10.00
8.00
4.00
2.00
1.00
0.00
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
Number of Processors
Parallel Portion
50%
75%
90%
95%

JVM ...

- ImageJ does its own internal work split and distribution
- Java does its own thread management
  - And garbage collection
  - And who knows what else
- So ...

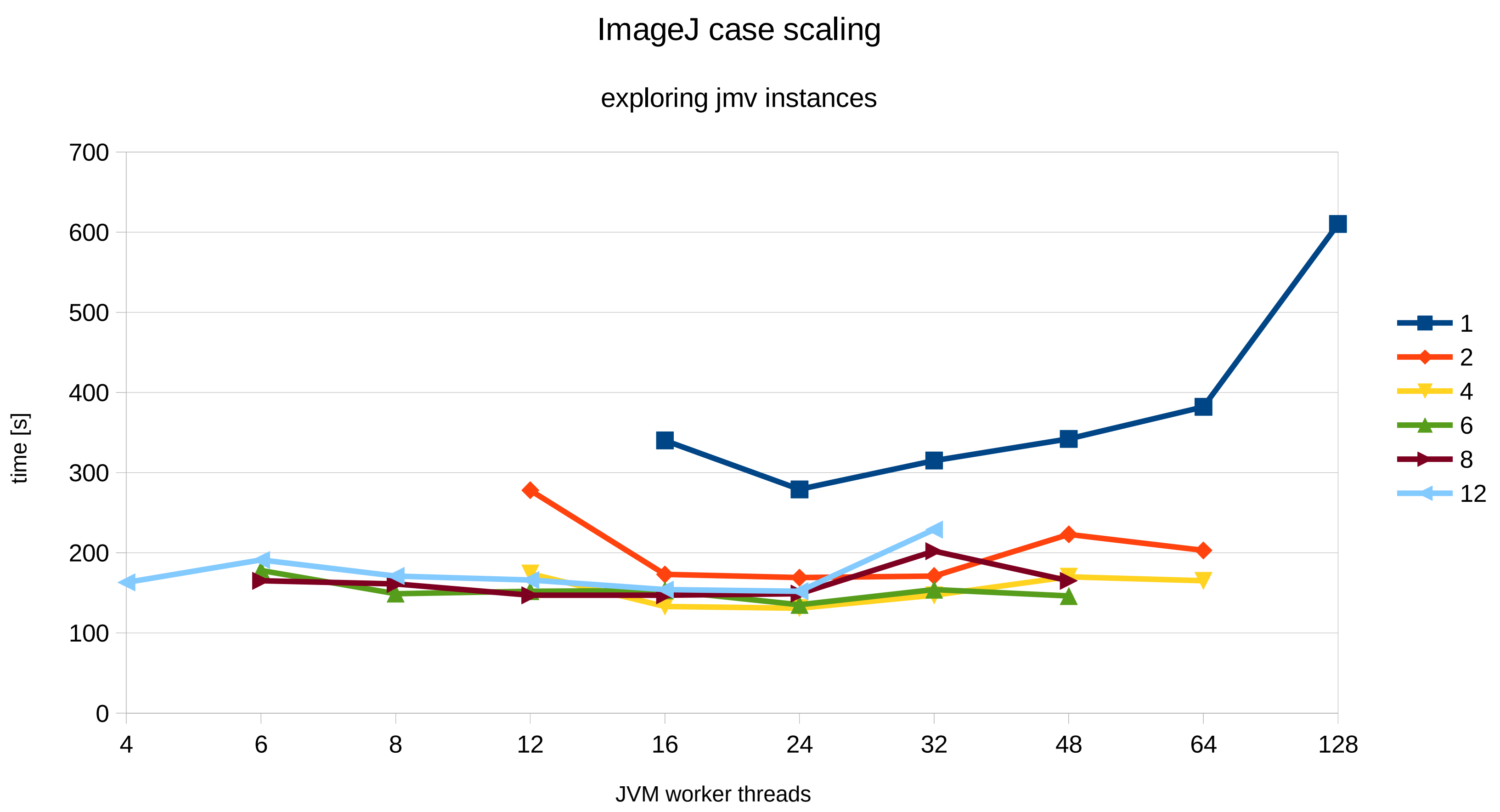
Lets overcommit number of java threads

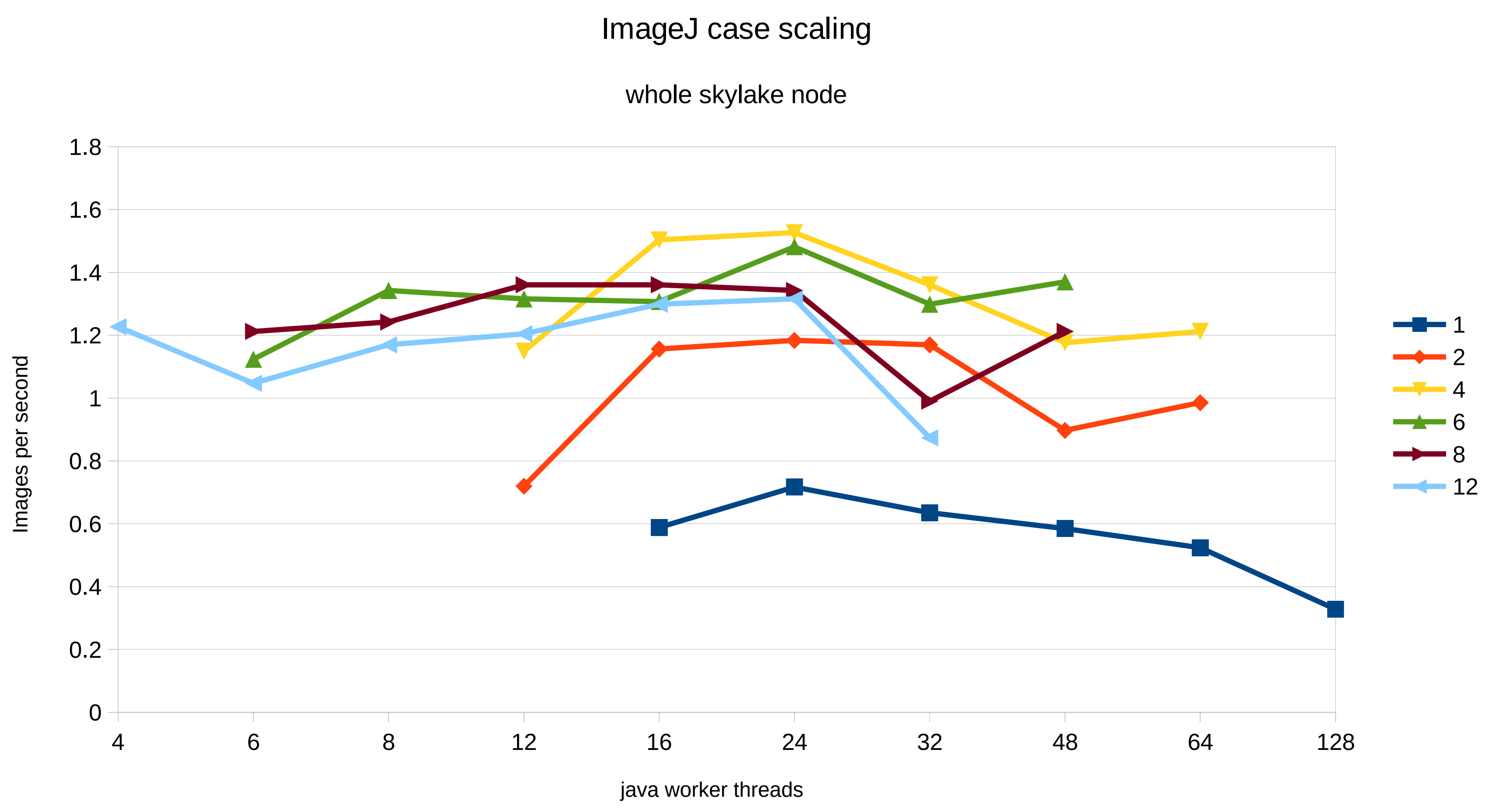


Can we do better?

- Try overlaping serial codepaths
- Try avoiding jvm internal overhead

- Split work across many jvm instances


ImageJ case scaling
exploring jmv instances
time [s]
700
600
500
400
300
200
100
0
4
6
8
12
16
24
32
48
64
128
JVM worker threads
1
2
4
6
8
12


ImageJ case scaling
whole skylake node
Images per second
java worker threads
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
4
6
8
12
16
24
32
48
64
128
1
2
4
6
8
12

Takeout

- JVM by default is not really high performance
  - It can be made to behave with enough effort
- Easiest method is to split work across many jvm instances
- And you get better throughput

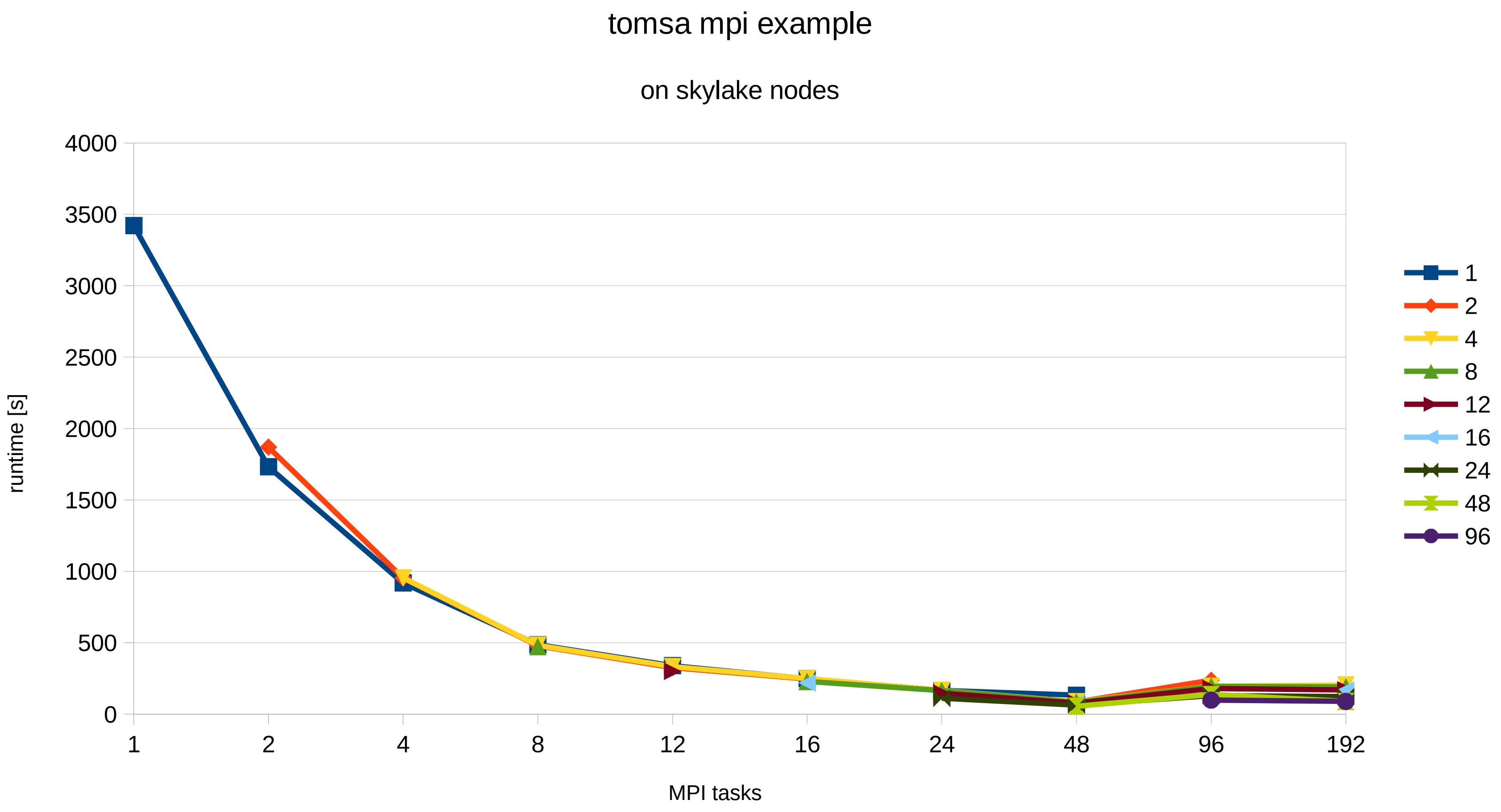
#SBATCH -n or -c ?

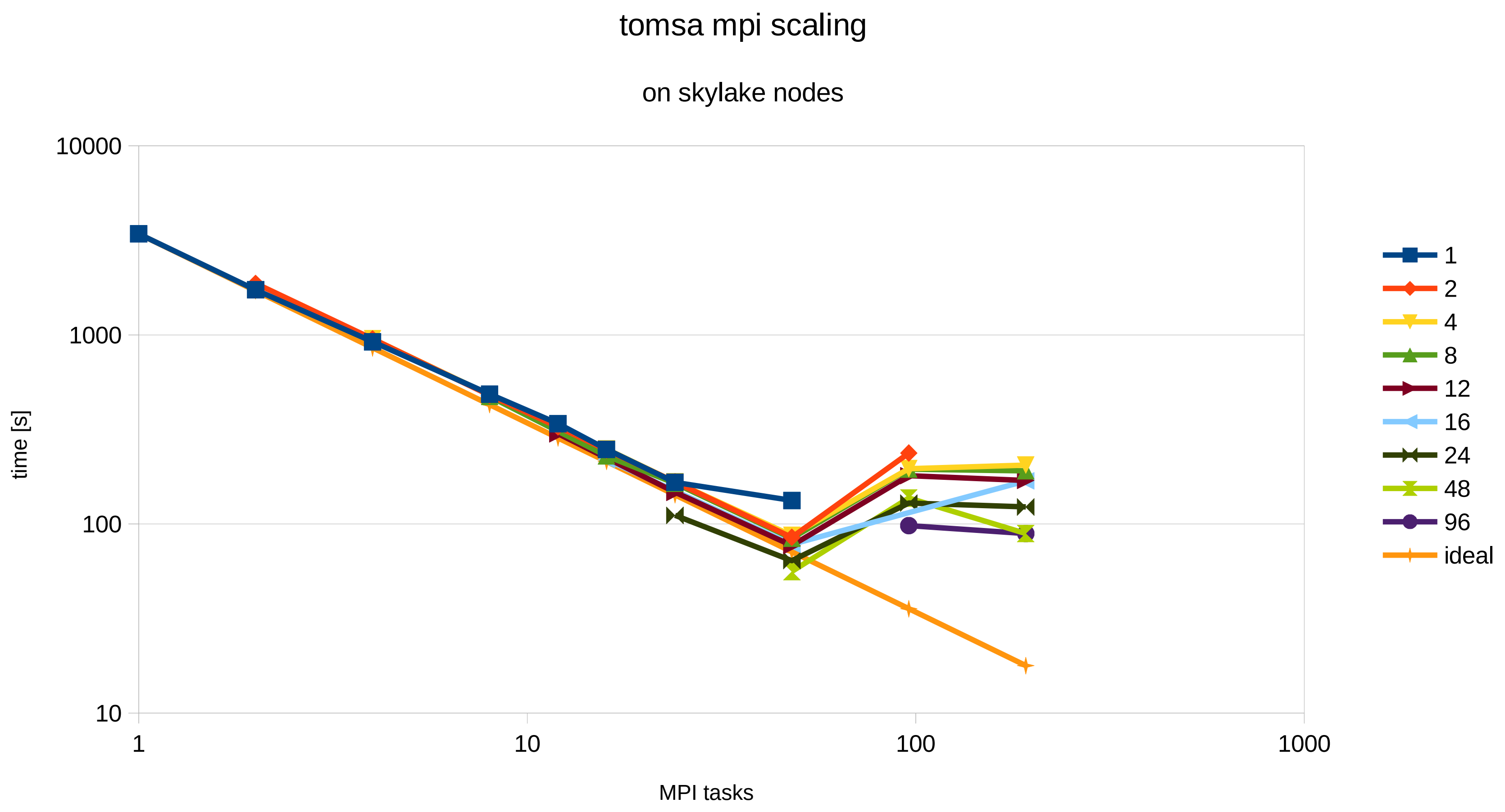
- -n is number of tasks
- -c is number of cores per task
- Each will tell slurm to give your job that number of cores
- Using both will tell slurm to give you n*c number of cores
- -n also instructs MPI how many ranks it can use
- -c also affects OMP_NUM_THREADS

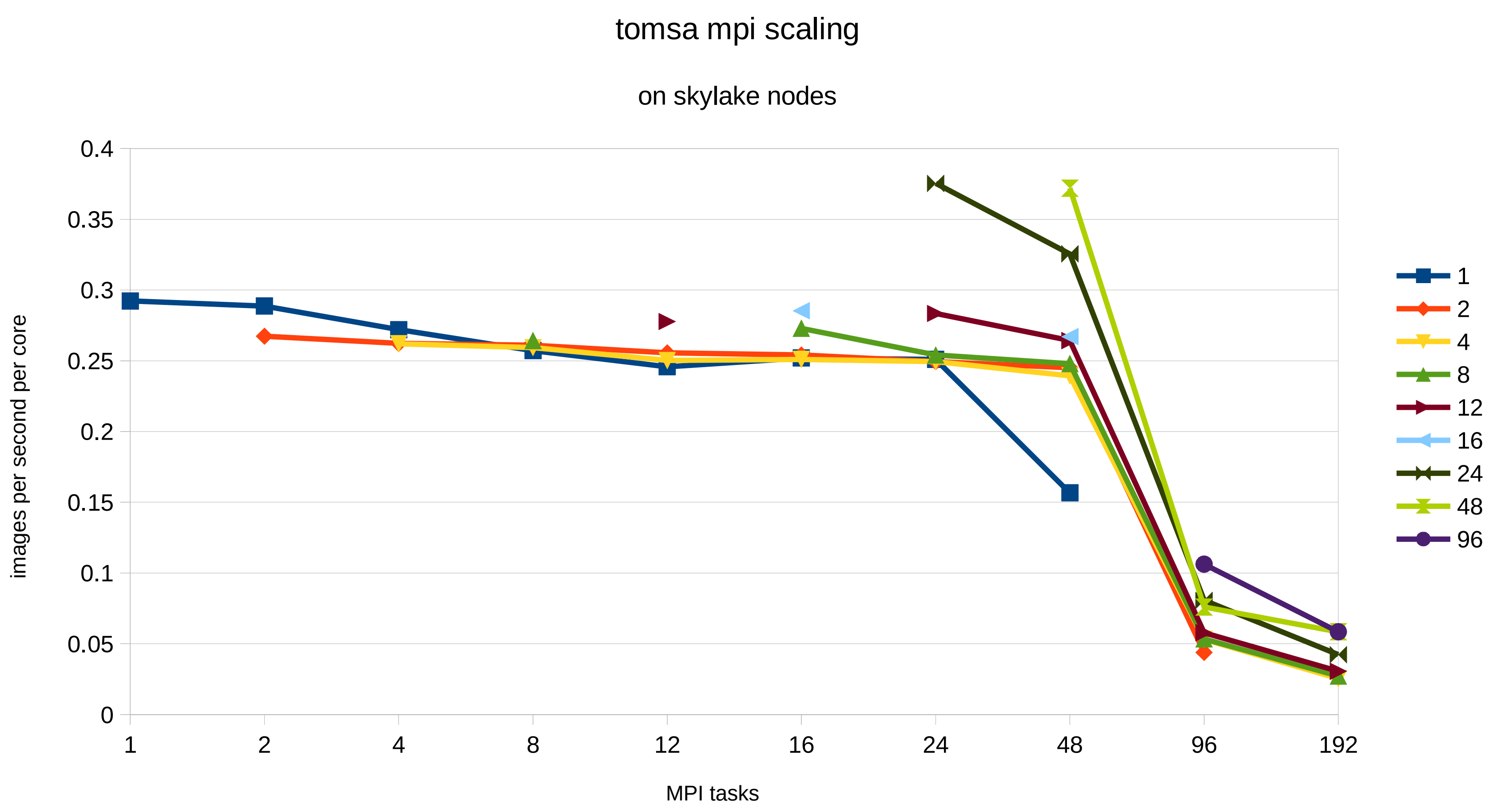
MPI example: tomsa

- Developed and provided by @turonova
- Workflow:
  - Extracts particles from tomograms
  - Averages all to create a reference
  - Runs few iterations of alignment
  - After best is found, averages again
  - Writes out new references and rotations
- See excercies/tomsa/run.sh

```
#!/bin/bash
#SBATCH -J tomsa_mpi
#also try -N with --ntasks-per-node
#SBATCH -n 24
#SBATCH -C "net10G|net25G"
#SBATCH --switches=1
#SBATCH –mem-per-cpu=20
#SBATCH --time=00:05:00

module load foss/2017b

cd $SCRATCHDIR
tar zxf /g/its/home/pecar/benchmarks/turonova_tomsa/data.tgz

START=$SECONDS
mpirun tomsa -param params.txt -folder ./results
echo took $((SECONDS-START)) seconds witn $SLURM_NTASKS tasks across
$SLURM_NNODES nodes

#cp results/* $SLURM_SUBMIT_DIR/
```


tomsa mpi example
on skylake nodes
runtime [s]
0
500
1000
1500
2000
2500
3000
3500
4000
1
2
4
8
12
16
24
48
96
192
MPI tasks
1
2
4
8
12
16
24
48
96


tomsa mpi scaling
on skylake nodes
time [s]
10000
1000
100
10
1
10
100
1000
MPI tasks
1
2
4
8
12
16
24
48
96
ideal


tomsa mpi scaling
on skylake nodes
images per second per core
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
4
8
12
16
24
48
96
192
MPI tasks
1
2
4
8
12
16
24
48
96

MPI takeouts

- Understand ratio of communication to computation of your code
- Always good idea to try to keep tasks as close as possible (from communication perspective)
  - Single node
  - #SBATCH --switches=1

Manual data level parallelism

How SLURM can help you

- Within a job
  - Use --ntasks-per-node and --cpus-per-task
  - Then use srun within job script to distribute work
- Collection of jobs
  - Array jobs

Within a job

- See examples/imagej/manytask.job

```
#!/bin/bash
#SBATCH -n 12 -c 4 --ntasks-per-node=1 --mem 16000 -t 0-00:10:00
module load Java
module load X11
mkdir -p ~/.imagej
cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/IJ_Prefs.txt ~/.imagej/
srun -n 12 --ntasks-per-node=1 cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/fib-sem--cell--8x8x8nm.tif $TMPDIR/
srun -n 12 --ntasks-per-node=1 cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/bg-er.classifier $TMPDIR/
cd $TMPDIR
START_TIME=$SECONDS
srun -n 1 /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,0,16,0,0',numWorkers='12'" &
srun -n 1 /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,17,32,0,0',numWorkers='12'" &
srun -n 1 /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,33,48,0,0',numWorkers='12'" &
srun -n 1  /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,49,64,0,0',numWorkers='12'" &
srun -n 1 /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,65,80,0,0',numWorkers='12'" &
srun -n 1  /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,81,96,0,0',numWorkers='12'" &
srun -n 1  /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,97,104,0,0',numWorkers='12'" &
srun -n 1  /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
8x8x8nm',outputModality='Save class probabilities as Tiff slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,105,120,0,0',numWorkers='12'" &
srun -n 1  /g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run  "Apply Classifier" "inputImageFile='fib-
sem--cell--8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-er.classifier',dataSetID='fib-sem--cell--
```

Array jobs

- Very useful if you have your data organized in some sequence
- Submit with --array=1-20 for 20 jobs
- See examples/imagej/array.job

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -c 4
#SBATCH --mem 4000
#SBATCH -t 0-00:03:00
#SBATCH --array=0-191:10%5

module load Java
module load X11

mkdir -p ~/.imagej
cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/IJ_Prefs.txt ~/.imagej/

cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/fib-sem--cell--8x8x8nm.tif $TMPDIR/
cp /g/its/home/pecar/benchmarks/tischer_fib-sem-cell-crop/bg-er.classifier $TMPDIR/

cd $TMPDIR
START_TIME=$SECONDS

/g/almf/software/Fiji.app/ImageJ-linux64 --mem=16000M --ij2 --allow-multiple --headless --run
"Apply Classifier" "inputImageFile='fib-sem--cell--
8x8x8nm.tif',memoryMB='32000',quitAfterRun='true',classifierFile='bg-
er.classifier',dataSetID='fib-sem--cell--8x8x8nm',outputModality='Save class probabilities as Tiff
slices',outputDirectory='.',inputModality='Open using ImageJ1
virtual',saveResultsTable='false',classificationIntervalXYZT='0,496,0,516,"$SLURM_ARRAY_TASK_ID","
$((SLURM_ARRAY_TASK_ID+10))",0,0',numWorkers='8'"

ELAPSED_TIME=$(($SECONDS - $START_TIME))

echo "Array $SLURM_ARRAY_TASK_ID took $ELAPSED_TIME seconds."
```

Organizing work from outside of cluster

- Python wiki page lists 30+ projects
- Language agnostic ones can be used with R, Perl, …
- Some also support different backends, such as cloud infrastructures
- R has package batchtools

In-house experience with:

- GC3Pie @pecar
- Drmaa @ralves
- Jug @coelho
- Snakemake @carnold
- Toil (CWL) @kbreuer

Q & A

Thanks
&
Happy Computing