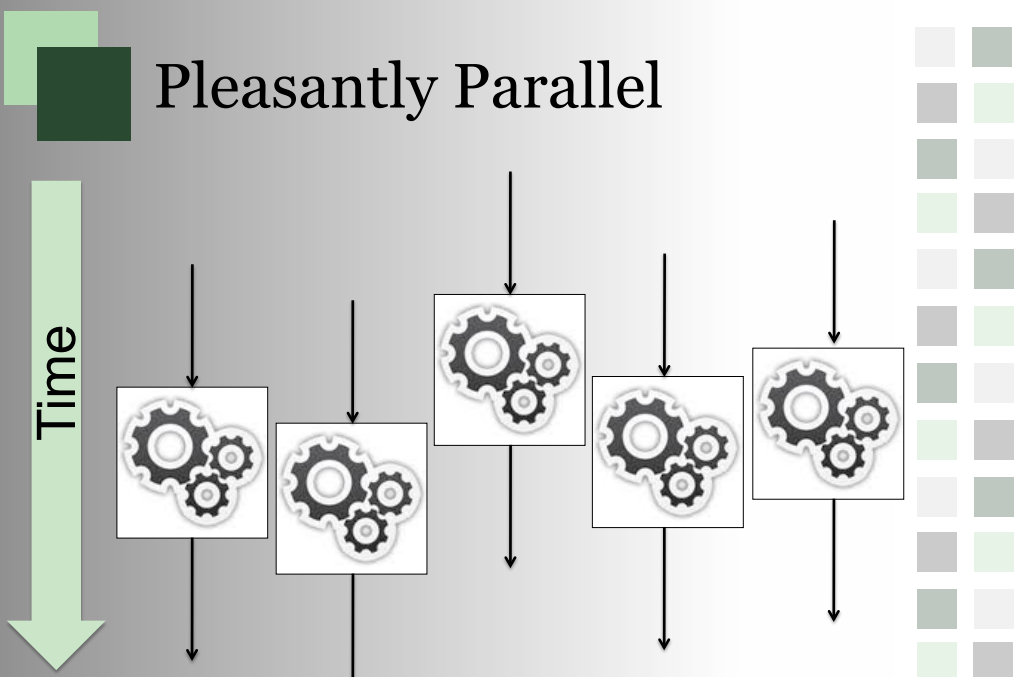


# Pleasantly Parallel

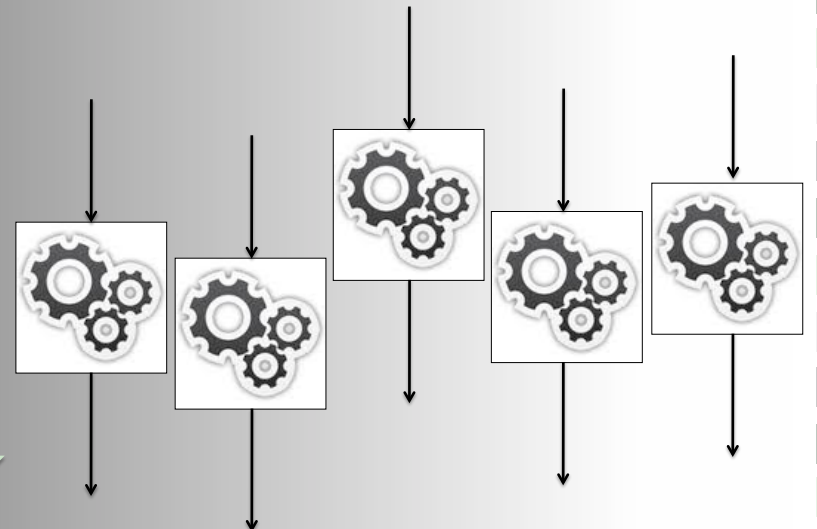
Dirk Colbry  
colbrydi@msu.edu  
Research Specialist  
Institute for Cyber-Enabled Research

MICHIGAN STATE UNIVERSITY  
© 2012 Michigan State University Board of Trustees.  
ICER



# Pleasantly Parallel

Time



MICHIGAN STATE UNIVERSITY  
ICER

## How fast can we go?

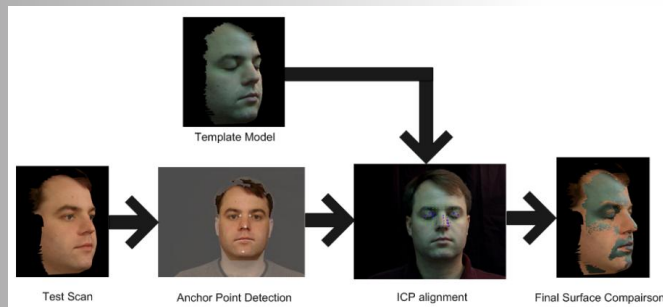
- T - How long does each operation take?
- N - How many operations do you need to run?
- CPUs – Number of Cores job will run on.
- Single CPU time estimate:
  - $T \times N$
- Best possible Pleasantly parallel time:
  - $(T \times N) * \text{overhead} / \text{CPUs}$

## Who are you? -- Biometrics



## Pairwise-All Problem

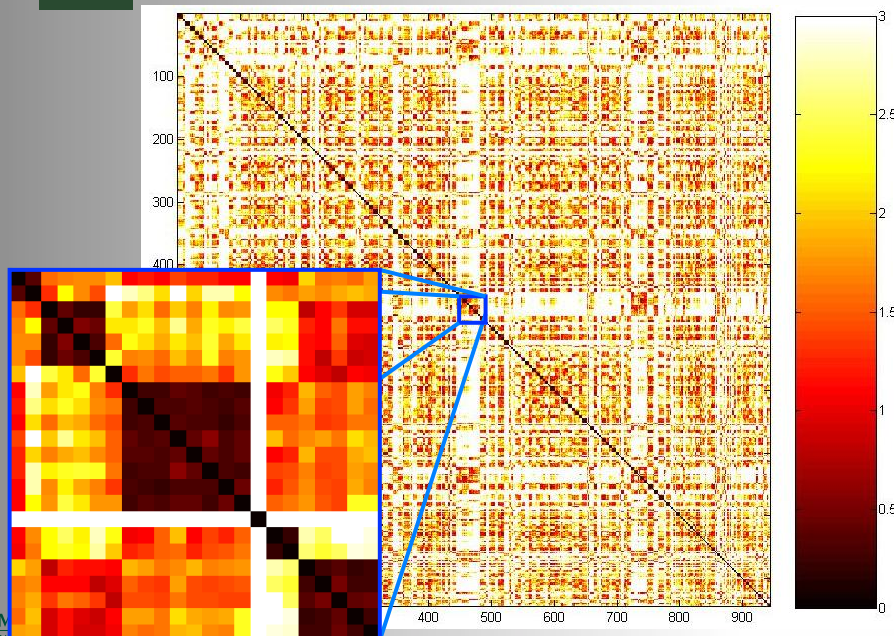
- Database of faces
- Compare everything to everything else
- Calculate a Matching score to use for identification



MICHIGAN STATE  
UNIVERSITY



## 943 x 943 Similarity Matrix



M

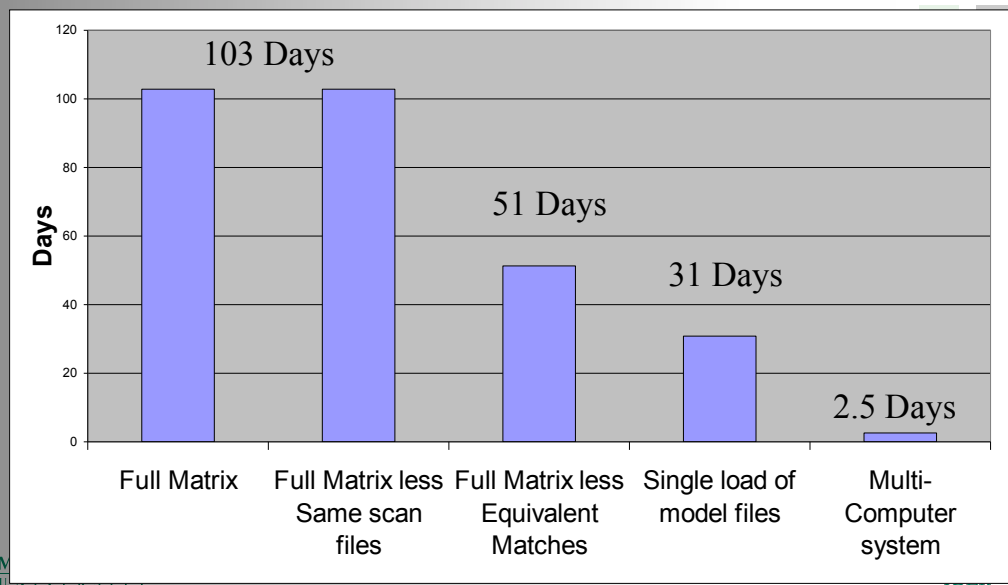
U



## Estimated Calculation Times

- Preprocessing
  - $943 * 12$  (seconds)  $\approx$  189 Minutes
- Matching
  - $943 * 943 * 5$  (seconds)  $\approx$  103 Days
- Scans matched to themselves always result in 0 mm
  - $(943 * 943 - 943) * 5$  (seconds)  $\approx$  103 Days
- The Proposed Alignment Algorithm is symmetric.
  - $(943 * 943 - 943)/2 * 5$  (seconds)  $\approx$  51.5 Days
- We also load models once per row instead of every time
  - $(943*943-943)/2 * 3$  (seconds) +  $943 * 2$  (seconds)  $\approx$  31 Days

## Calculation Time for Full Similarity Matrix



## How do we go even bigger?

- 5000 scans.
  - 1.5 years on a single processor computer
  - 13 days on our ad-hoc cluster.
  - 1.5 days a commodity cluster at MSU

## Step to Pleasantly Parallel

- Figure out command line
- Estimate single job time:
  - Should be > 5 minutes
  - Should be < 1 week
  - Best if < 4 hours
- Make a submissions script
- Submit Job

## Example

- Folder full of input files:

1.in	5.in	9.in	13.in	17.in
2.in	6.in	10.in	14.in	18.in
3.in	7.in	11.in	15.in	19.in
4.in	8.in	12.in	16.in	

- Want folder full of output files:

1.out	5.out	9.out	13.out	17.out
2.out	6.out	10.out	14.out	18.out
3.out	7.out	11.out	15.out	19.out
4.out	8.out	12.out	16.out	

- Command Syntax:

– `./myprogram inputfile > outputfile`

## PBS Job Arrays

- One submission script copied many times
- Uses the PBS `-t` option
  - Ranges: 1-10
  - Lists: 2,4,100,3
  - Combination: 1-10,20,50,100
- Distinguish between jobs by using the `PBS_ARRAYID` environment variable

## Simple Job Array

```
#!/bin/bash -login
#PBS -l walltime=00:05:00,mem=2gb
#PBS -l nodes=1:ppn=1,feature=gbe
#PBS -t 1-100

cd ${PBS_O_WORKID}

./myprogram ${PBS_ARRAYID}.in > ${PBS_ARRAYID}.out

qstat -f ${PBS_JOBID}
```

## Example: Job Arrays

- Get the blender\_farm example:
 

```
> getexample blender_farm
```

```
> cd ./blender_farm
```
- Look at the qsub file, using “less” command
 

```
> less blender_farm.qsub
```
- Submit the job
 

```
> qsub blender_farm.qsub
```

## HPCC Job array limitations

- Can not have more than 144 cores running at once
- Can not submit more than 256 jobs at once (This limitation should change soon).

## Job array numbers

- All numbers in a job array have the same base number
  - 7478210
- Each PBS\_ARRAYID is show in square brackets
  - 7478210[1]
  - 7478210[2]
- Delete all jobs using one command
  - qdel 7478210[]



## Unrolling Loops

- Your program has independent loops
  - Each iteration of the loop does not depend on the other iterations
  - Loop can be executed in any order
  - 5 Minutes < Iteration Time < 1 week
  - Output of each iteration must be easy to save and recombine for next step of workflow
- Rewrite your program to accept an iteration number as an input
  - ./myprogram IterationNumber
- Rewrite your program to save output and use an additional program for post processing

## Simple Unrolled Loop

```
#!/bin/bash -login
#PBS -l walltime=00:05:00
#PBS -l nodes=1:ppn=1,feature=gbe
#PBS -t 1-100

cd ${PBS_O_WORKID}

./myprogram ${PBS_ARRAYID}

qstat -f ${PBS_JOBID}
```

## Task Queue

- A list of tasks (treatments, inputs, ...) that distinguish what needs to be done.
- Each pleasantly parallel process (worker) checks the list and picks work not completed yet.
- The trick is to not have two workers do the same task.

## Files as Semaphores (FAS)

- Use a list of input files as your task list
- Use a list of output files (or flag files) as your in-progress/complete list
- Rely on the file system to ensure that no two jobs are selected at the same time (not a great assumption but it works)

## Simple FAS

```
#!/bin/bash -login
#PBS -l walltime=00:05:00
#PBS -l nodes=1:ppn=1,feature=gbe
#PBS -t 1-100
cd ${PBS_O_WORKID}
sleep $(( ${RANDOM} % 100 ))

for file in *.in; do
    output="./${file%.*}.out"
    if [ ! -f ${output} ]; then
        touch ${output}
        ./myprogram ${file} > ${output}
        qsub -t 0 -N ${PBS_JOBNAME} ${0}
    fi
done
```

MICHIGAN  
UNIVERSITY



## List of Commands

- Commands.txt

```
./myprogram -a 100 -z 3023
./myprogram dosomething different
./myprogram
./myprogram -s 100
./myprogram -s 200
./myprogram -s 300
./myprogram -w 400
./myotherporgram
./mythirdprogram
```

MICHIGAN STATE  
UNIVERSITY



## List of Commands



```
#!/bin/bash -login
#PBS -l walltime=00:05:00
#PBS -l nodes=1:ppn=1,feature=gbe
#PBS -t 1-100

cd ${PBS_O_WORKID}

cmd=`tail -n ${PBS_ARRAYID} commands.txt | head -n 1`
echo ${cmd}
${cmd}

qstat -f ${PBS_JOBID}
```



## Condor High Throughput Computing

- Job submission system
- Runs like a screen saver
- Steals CPU Cycles

