Cluster architectures and Calculations

Assignment 3

Prototein Folding

Tzemis Evangelos

khb107@alumni.ku.dk

Modification of the sequential code was done in the following way. Prototein.py calls the folding function contained in calcfolding_pastSet.py. This is responsible for spawning the new processes and creating the appropriate element inside the tuple space. Spawned processes are executing the calcWorker.py file which is responsible for finishing the mapping and maintaining the best score of the prototein folding. There is a decomposition figure at the end of this file.

Enter – Halting Pastset

Prototein.py is responsible for entering and halting the pastSet. The initialization time is not included in the time measurements, which is obvious from the relevant code. Furthermore we have to pass the number of workers and the protein that it is going to work as arguments from the command line.

```
pset = pastset.PastSet()
start=time.time()
(bestscore, winner) = folding(pset,p, best)
stop=time.time()
print 'Length ',len(p),'took',stop-start,'sec'
print 'BestScore was ', bestscore
pset.halt()
```

Master – calc_folding_pastSet.py

Folding function enters all elements in the tuple space. We use the following elements.

- Results (int, list) : contains the best result of each individual worker. To find the best we need to compare them and issue the best score.
- Protein-jobs (list,list) : when we reach the appropriate analysis depth we put the (map, place) lists in the tuplespace, in order for workers to continue the calculations.
- Protein (list,) : Used by the workers and contains only the initial protein.

First we use pastset.enter() to gain access to the above elements, and then we spawn #cpu workers

```
ProtBest = pset.enter(('Result', int,list))
ProtJobs = pset.enter(('Protein-jobs',list,list)) # map, place
ProtSeries = pset.enter(('Protein',list))   # protein
'''insert the protein inside the tuple'''
ProtSeries.move((protein,))
'''create all the workers'''
for i in xrange(workers):
    pset.spawn('a3/calcWorker.py', '')
```

Next comes calling to fold function, which puts the semi-folded proteins in the tuple space. It does that when they reach the length of 8. It returns the numbers of elements (NE) she placed in the tuple space. Finally we wait to observe NE elements form the Result element and we do the comparison to specify the best score among them.

```
counter = fold(protein, mapp, place, ProtJobs,0)
best = -9999999
for i in xrange(counter):
    (bestscore, winner) = ProtBest.observe()
    if bestscore > best:
      best = bestscore
```

Worker – calcWorker.py

Worker in turn initializes past set, and then for every job it observes from the tuple space it calls folWorker which returns the local best for the current map and place lists. Then it places is to the Result element and tries to read another job, if any.

```
pset = pastset.PastSet()
while True:
  protein = ProtSeries.observe(ProtSeries.last()-1)[0]
  (map,place) = ProtJobs.observe()
  (best, winner) = foldWorker(protein, map, place,(-9999999,[]))
  ProtBest.move((best,winner))
```

In general jobs are more than the available workers which means that many of them (if not all) will have to work for more than one jobs.

Result and Performance Graphs

In the horizontal axis we have the number of CPU's. For each of them we have three columns indicating the three different protein length (17, 18 and 19).

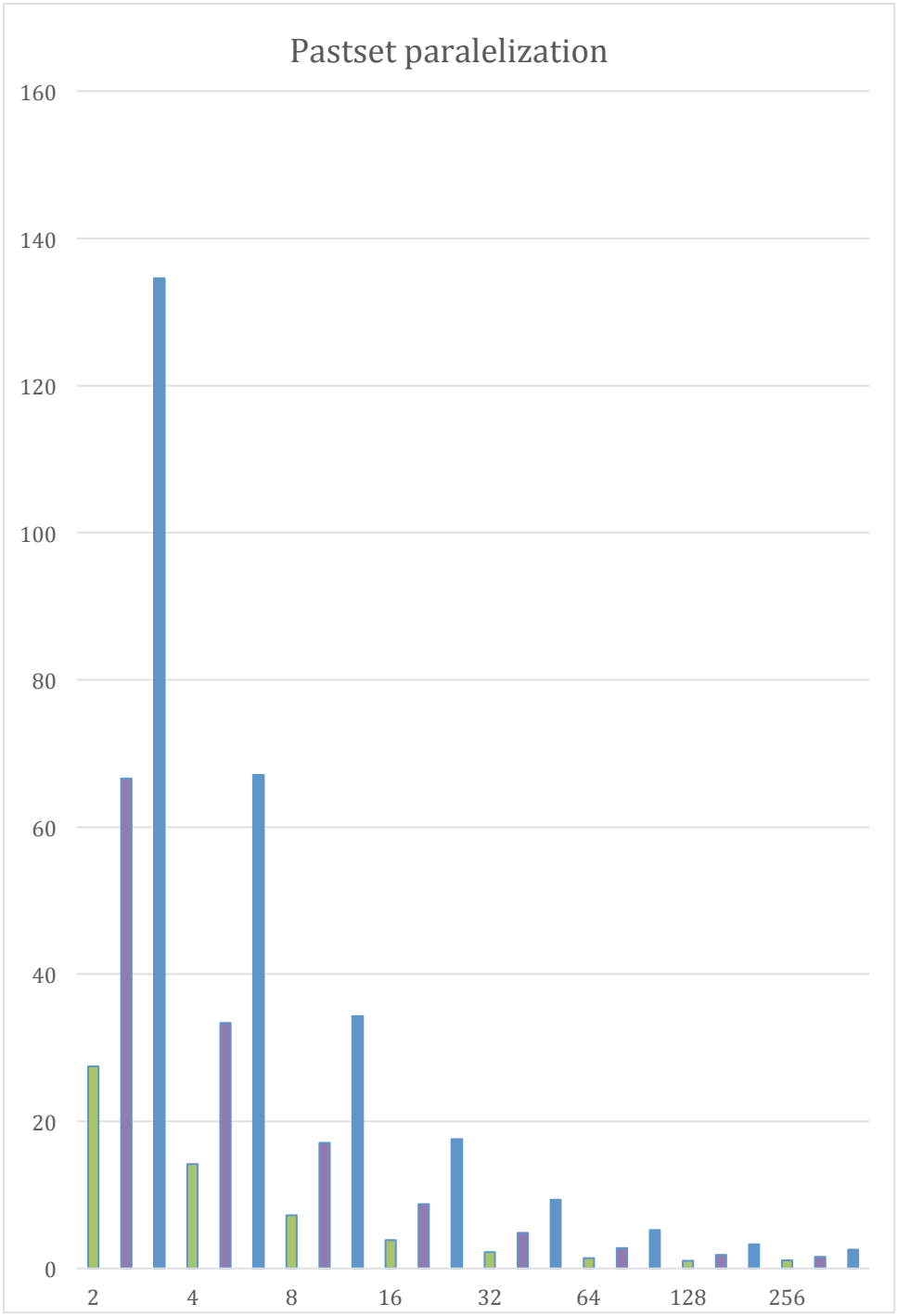

Figure 1 – Runtime - green = length 17 purple = length 18 blue = length 19

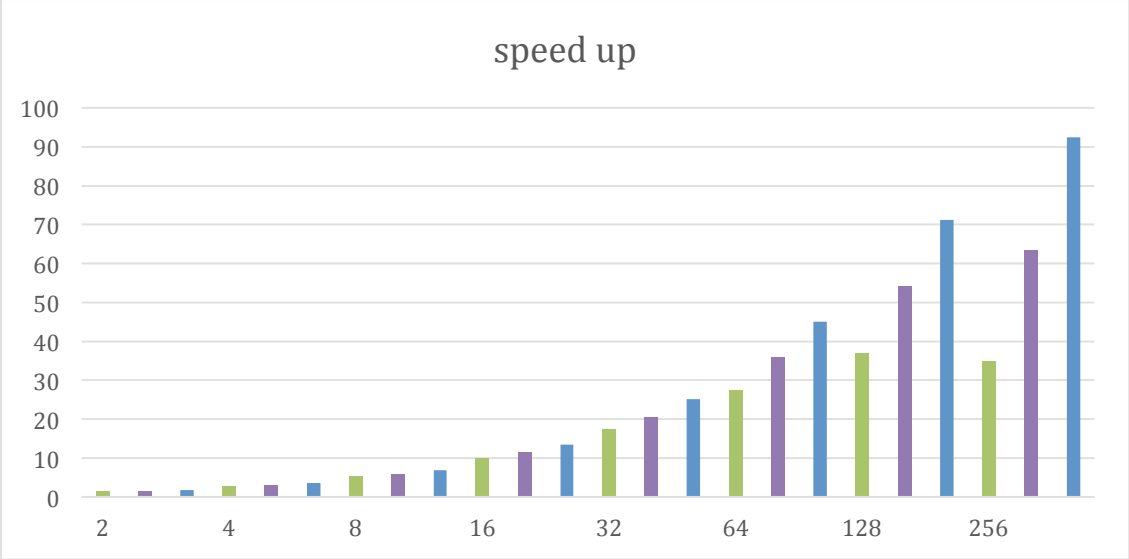

Figure 2 - Speed up green = length 17 purple = length 18 blue = length 19