A Moldable Online Scheduling Algorithm and Its Application to Parallel Short Sequence Mapping

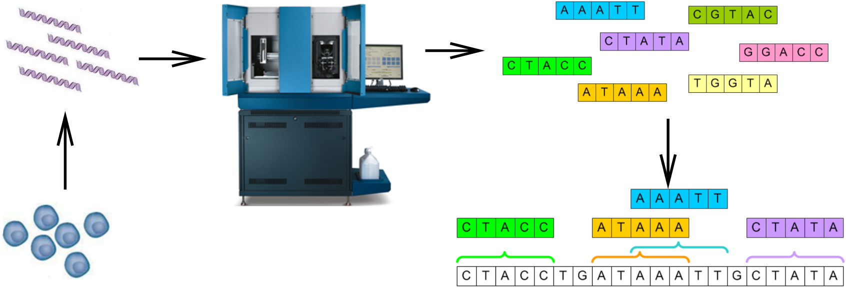
Erik Saule, Doruk Bozdağ, Umit V. Catalyurek

Department of Biomedical Informatics, The Ohio State University
{esaule,bozdagd,umit}@bmi.osu.edu

JSSPP 2010

Supported by the U.S. DOE SciDAC Institute, the U.S. National Science Foundation and the Ohio Supercomputing Center

Motivation



Sequencing

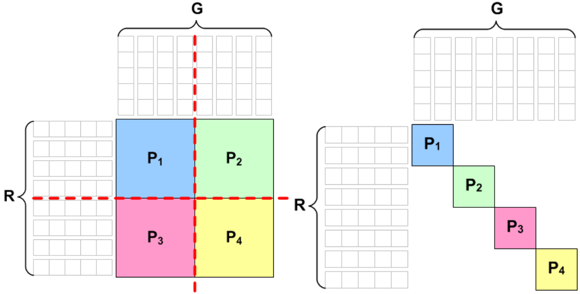
- Next generation sequencing instruments (SOLiD, Solexa, 454) can sequence up to 1 billion bases a day
  - Hundreds of millions of 35-50 base reads

Mapping

- Map reads to a reference genome efficiently (Human genome: 3Gb)
- Need large parallel computer
- Pooling resource will decrease cost
- We study the job scheduling problem

Parallel Short Sequence Mapping[Bozdag *et al.*, IPDPS 09]

Three partitioning dimensions:



$$P(m_g, m_r, m_s) = c_{gs}\frac{G}{m_g} + c_g\frac{G}{m_g m_s} + c_{rs}\frac{R}{m_r} + (c_r + c_c\frac{G}{m_g m_s})\frac{R}{m_r m_s}$$

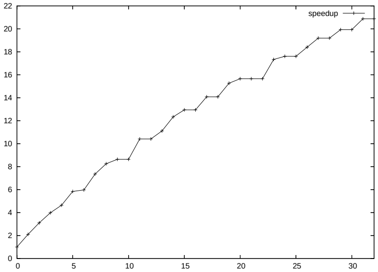
Partitioning on m processors is finding minimum $P(m_g, m_r, m_s)$ such that $m_g m_r m_s \leq m$

Outline of the Talk

Parallel Short Sequence Mapping

The important facts:

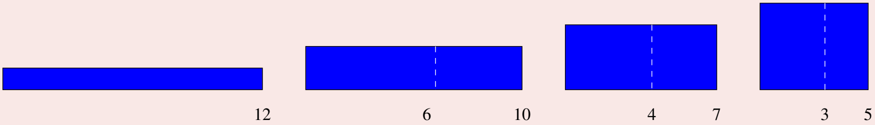
- can adapt to different number of processor
- good runtime prediction function
- no super linear speed up
- non convex speedup function (steps)
- no preemption



Moldable Scheduling

Instance

- m processors
- n tasks
- Task i arrives at r_i
- The execution of i on j processors takes $p_{i,j}$ time units



Solution

- Task i is executed on π_i processors
- Task i starts at σ_i
- Task i finishes at $C_i = \sigma_i + p_{i,\pi_i}$

Objective Function

Flow time

The flow time is the time spent in the system per a task $F_i = C_i - r_i$.

- Does not take task size into account.
- Optimizing the maximum flow time is unfair to small tasks.
- Optimizing the average flow time should starve large tasks.

Stretch [Bender *et al.* SoDA 98]

The stretch is the flow time normalized by the processing time of the task. In the moldable tasks context, we define it as $s_i = \frac{C_i - r_i}{p_{i,1}}$.

- It provides a better fairness between tasks.
- Optimizing maximum stretch avoids starvation.

Objective Function

Flow time

The flow time is the time spent in the system per a task $F_i = C_i - r_i$.

- Does not take task size into account.
- Optimizing the maximum flow time is unfair to small tasks.
- Optimizing the average flow time should starve large tasks.

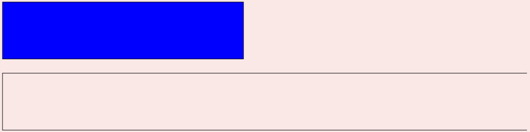
Stretch [Bender *et al.* SoDA 98]

The stretch is the flow time normalized by the processing time of the task. In the moldable tasks context, we define it as $s_i = \frac{C_i - r_i}{p_{i,1}}$.

- It provides a better fairness between tasks.
- Optimizing maximum stretch avoids starvation.

Online maximum stretch can not be approximated

Adversary technique on one processor



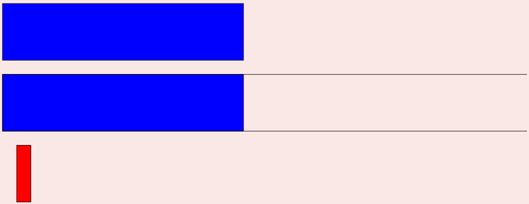
A large task enters in the system

On several processors

There are similar techniques on several processors but there are more complicated and thus less prone to appear in practice.
The key point: if all processors are busy, a small task entering the system will have a large stretch.

Online maximum stretch can not be approximated

Adversary technique on one processor



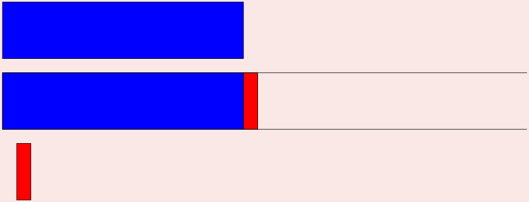
If it is scheduled immediately, a small task is sent

On several processors

There are similar techniques on several processors but there are more complicated and thus less prone to appear in practice.
The key point: if all processors are busy, a small task entering the system will have a large stretch.

Online maximum stretch can not be approximated

Adversary technique on one processor



It suffers a large delay (and an unbounded stretch)

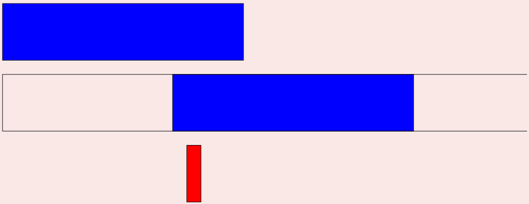
On several processors

There are similar techniques on several processors but there are more complicated and thus less prone to appear in practice.

The key point: if all processors are busy, a small task entering the system will have a large stretch.

Online maximum stretch can not be approximated

Adversary technique on one processor



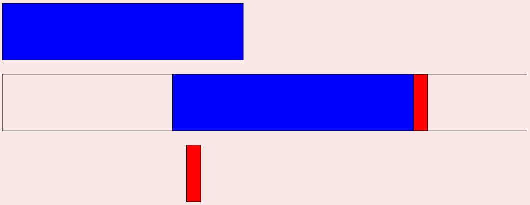
If the large task is scheduled later, a small task is sent accordingly

On several processors

There are similar techniques on several processors but there are more complicated and thus less prone to appear in practice.
The key point: if all processors are busy, a small task entering the system will have a large stretch.

Online maximum stretch can not be approximated

Adversary technique on one processor



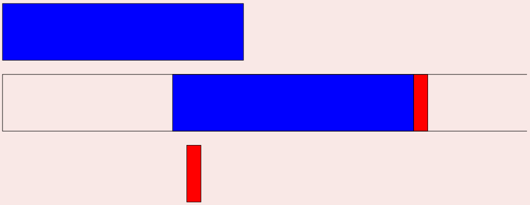
It suffers a large delay (and an unbounded stretch)

On several processors

There are similar techniques on several processors but there are more complicated and thus less prone to appear in practice.
The key point: if all processors are busy, a small task entering the system will have a large stretch.

Online maximum stretch can not be approximated

Adversary technique on one processor



It suffers a large delay (and an unbounded stretch)

On several processors

There are similar techniques on several processors but there are more complicated and thus less prone to appear in practice.
The key point: if all processors are busy, a small task entering the system will have a large stretch.

Outline of the Talk

Principle of the Deadline Based Online Scheduler (DBOS)

- All tasks running concurrently should get the same stretch to maximize efficiency
- Using the optimal maximum stretch as an instant measure of the load
- Aim at a more efficient schedule than the optimal instant maximum stretch one to deal with still-to-arrive tasks

The DBOS Algorithm

Targeting a maximum stretch S

Task i must complete before the deadline $D_i = r_i + p_{i,1}S$.

Moldable Earliest Deadline First (MEDF)

- Considers task in deadline order.
- Allocates the minimum number of processors to each task to completes before the deadline.
- Schedules the task as soon as possible without moving any other task.

$DBOS(\rho)$

- Estimate the best maximum stretch $S*$ using a binary search.
- The deadline problem is solved by MEDF.
- Build a schedule of good efficiency of stretch $\rho S*$.
  - ρ is the online parameter

The DBOS Algorithm

Targeting a maximum stretch S

Task i must complete before the deadline $D_i = r_i + p_{i,1}S$.

Moldable Earliest Deadline First (MEDF)

- Considers task in deadline order.
- Allocates the minimum number of processors to each task to completes before the deadline.
- Schedules the task as soon as possible without moving any other task.

$DBOS(\rho)$

- Estimate the best maximum stretch $S*$ using a binary search.
- The deadline problem is solved by MEDF.
- Build a schedule of good efficiency of stretch $\rho S*$.
  - ρ is the online parameter

The DBOS Algorithm

Targeting a maximum stretch S

Task i must complete before the deadline $D_i = r_i + p_{i,1}S$.

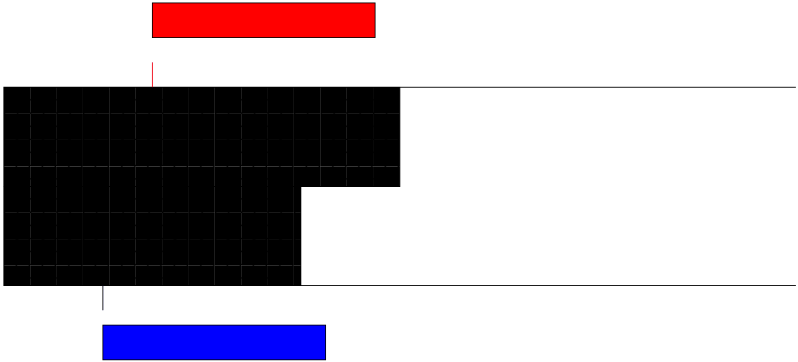
Moldable Earliest Deadline First (MEDF)

- Considers task in deadline order.
- Allocates the minimum number of processors to each task to completes before the deadline.
- Schedules the task as soon as possible without moving any other task.

$DBOS(\rho)$

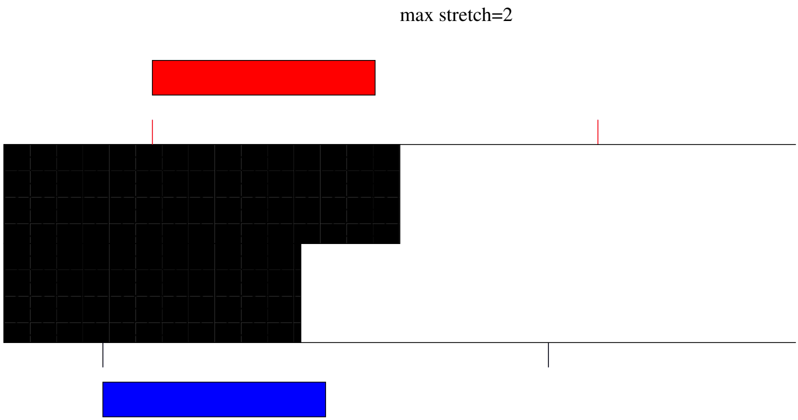
- Estimate the best maximum stretch $S*$ using a binary search.
- The deadline problem is solved by MEDF.
- Build a schedule of good efficiency of stretch $\rho S*$.
  - ρ is the online parameter

An example



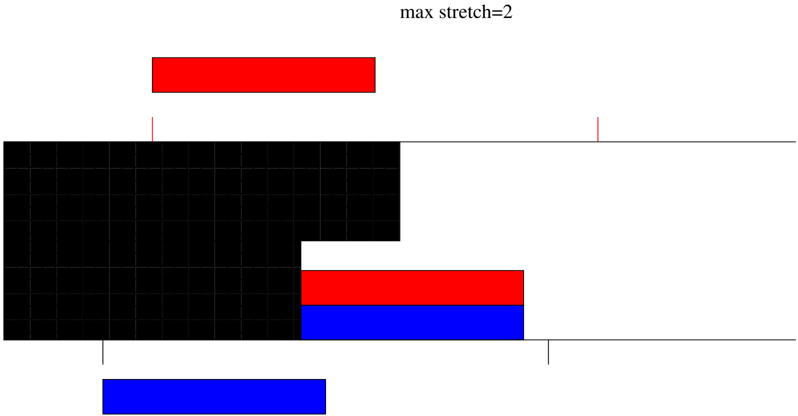
A system with two pending tasks

An example



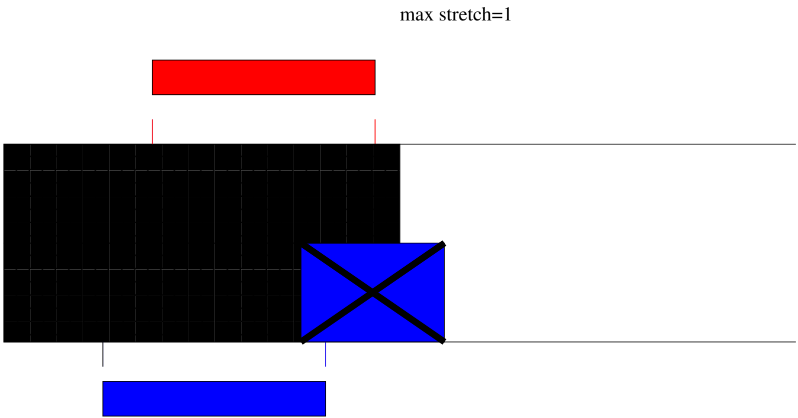
Deadlines induced by a stretch of 2

An example



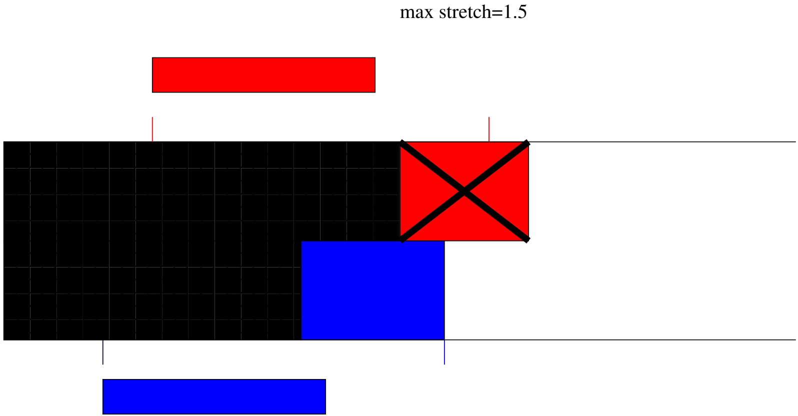
A maximum stretch of 2 is reachable

An example



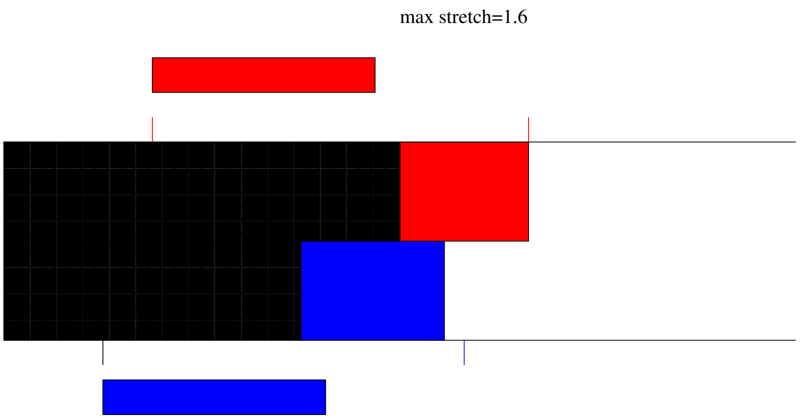
But 1 is not

An example



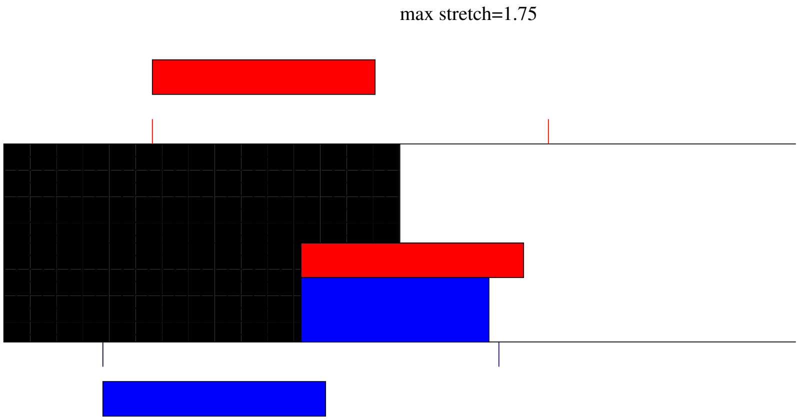
Neither 1.5

An example



The best stretch is 1.6

An example



The online parameter $\rho = 1.1$ leaves much more space (thanks to MEDF).

Outline of the Talk

An Iterative Process [Sabin et al, JSSPP 06]

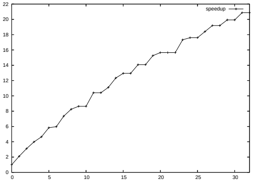
The algorithm

Processor allocation are evaluated using the flow-time of the FCFS schedule

- Starts with one processor per task.
- Try to add one processor to the task that will reduce its processing time the most
- If it is better, keep it
- Otherwise remove the processor and never try that task again

Properties

- Optimizing flow time
- Claimed to outperform fair share
- Parameter-less



Improvement

If the speedup function is non convex or has steps. The algorithm gets stuck.
Modification: step to the next point on the convex hull

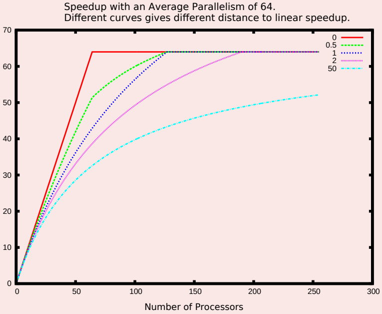
First Experimental Setting

Goal: assess performance on a well known setting

Downey model

Two parameters:

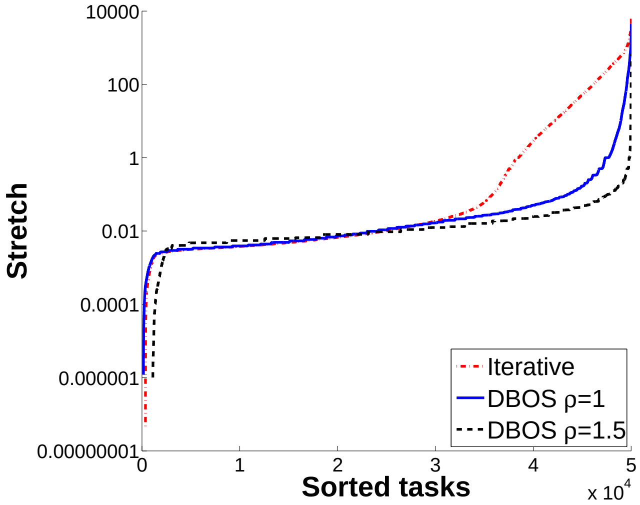
- Average parallelism
- Distance to linear speedup



Generation

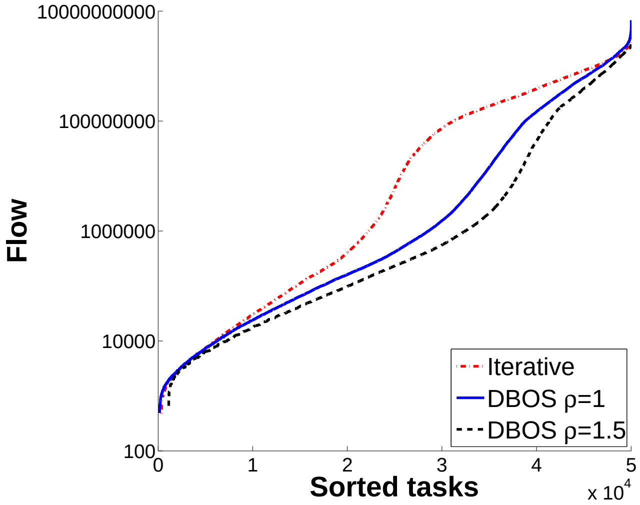
- 512 processors
- First 5000 tasks of SDSC Par 96 (From the Feitelson archive)
- Sequential time : total execution time
- Average parallelism : between number of used processor and 512
- Distance to linear speedup : between 0 and 2

Downey model results



DBOS generates less tasks with high stretch.

Downey model results



DBOS leads to better flow time. Iterative could be improved.

Second Experimental Setting

Goal: test case reflecting the cluster usage

Generation

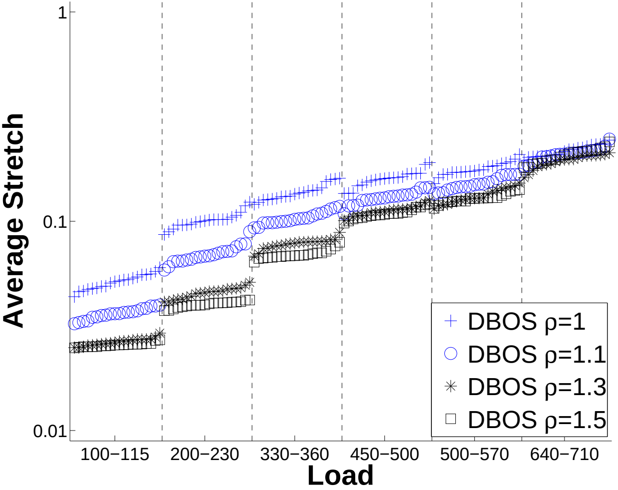
- 512 processors
- Each task corresponds to one lab studying one genome
- Speedup according to the runtime prediction function
- 5000 tasks with exponential inter-arrival time
- Changing the parameter of the exponential to control the load

Real data

Sequencing machine	Reads
454 GS FLX Genome Analyzer	1 million
Solexa IG sequencer	200 million
SOLiD system	400 million

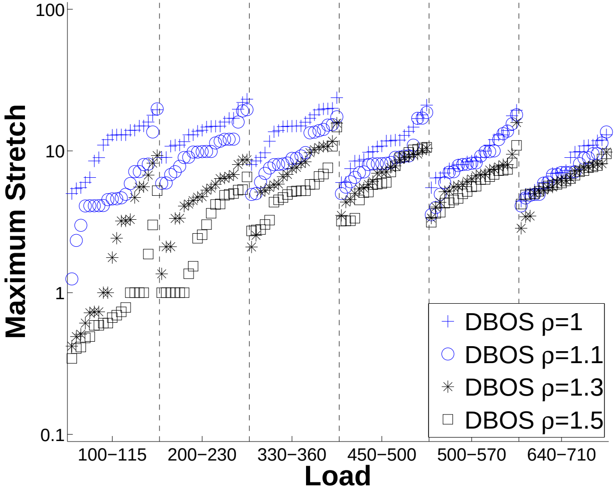
Genome	Size
E. Coli	4.6 million
Yeast	15 million
A. Thaliana	100 million
Mosquito	280 million
Rice	465 million
Chicken	1.2 billion
Human	3.4 billion

Mapping : the online parameter (average stretch)



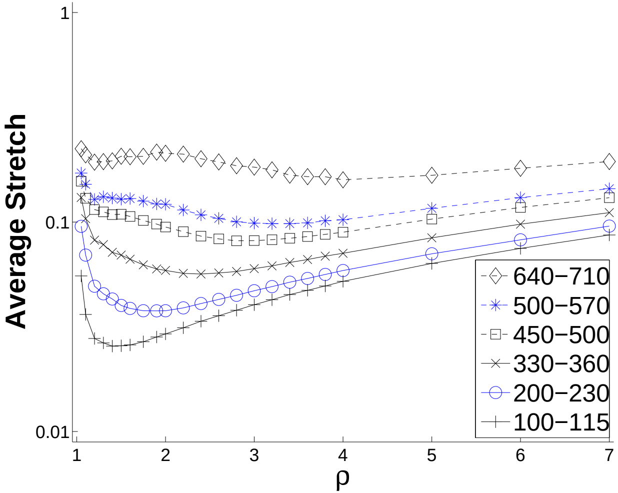
Quickly drops with ρ. Step at $\rho = 1.3$.

Mapping : the online parameter (maximum stretch)



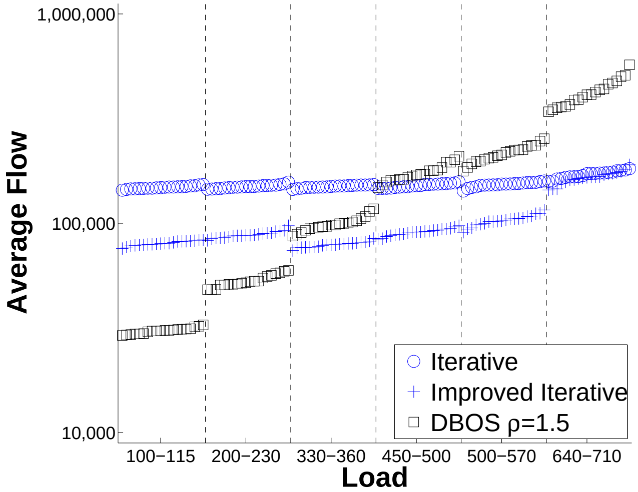
Max stretch is kept at a reasonable level. The online parameter ρ is very helpful here.

Mapping : tuning the online parameter



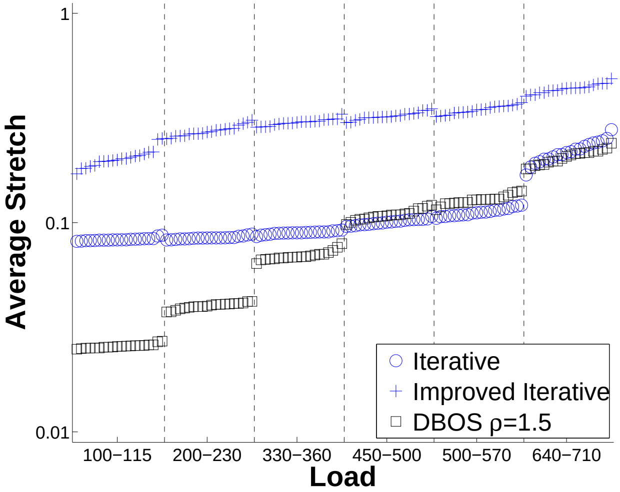
On non-overloaded cases, the average stretch is bimonotonic. A reasonable ρ value is easy to find.

Mapping : DBOS vs Iterative (average flow)



DBOS is competitive.

Mapping : DBOS vs Iterative (average stretch)



DBOS leads to much better stretch (even when iterative got stuck).

Outline of the Talk

The end

Conclusion

- Pooling the resources in short sequence mapping operation should lower the costs.
- To provide fairness stretch should be considered instead of flow time.
- An scheduling algorithm is proposed to optimize stretch and avoid worst case online scenario.
- Which performs well on Short Sequence Mapping application.

Perspective

- Investigate other ways to avoid worst case scenarios.
- Study more simple algorithms/models to get reference points.

The end

Conclusion

- Pooling the resources in short sequence mapping operation should lower the costs.
- To provide fairness stretch should be considered instead of flow time.
- An scheduling algorithm is proposed to optimize stretch and avoid worst case online scenario.
- Which performs well on Short Sequence Mapping application.

Perspective

- Investigate other ways to avoid worst case scenarios.
- Study more simple algorithms/models to get reference points.

A Moldable Online Scheduling Algorithm and Its Application to Parallel Short Sequence Mapping

Erik Saule, Doruk Bozdağ, Umit V. Catalyurek

Department of Biomedical Informatics, The Ohio State University
{esaule,bozdagd,umit}@bmi.osu.edu

JSSPP 2010

Supported by the U.S. DOE SciDAC Institute, the U.S. National Science Foundation and the Ohio Supercomputing Center