Conquering Massive Clinical Models with GPU Parallelized Logistic Regression

Yuxi Tian

OHDSI PLE Workgroup
June 7, 2018

Introduction

- Observational data provides an enormity of information
- Using an enormity of information requires massive statistical models
- Fitting massive models is computationally prohibitive (especially with cross-validation)

What can we do?

- Exploit graphics processing units for massive parallelization potential

Logistic Regression

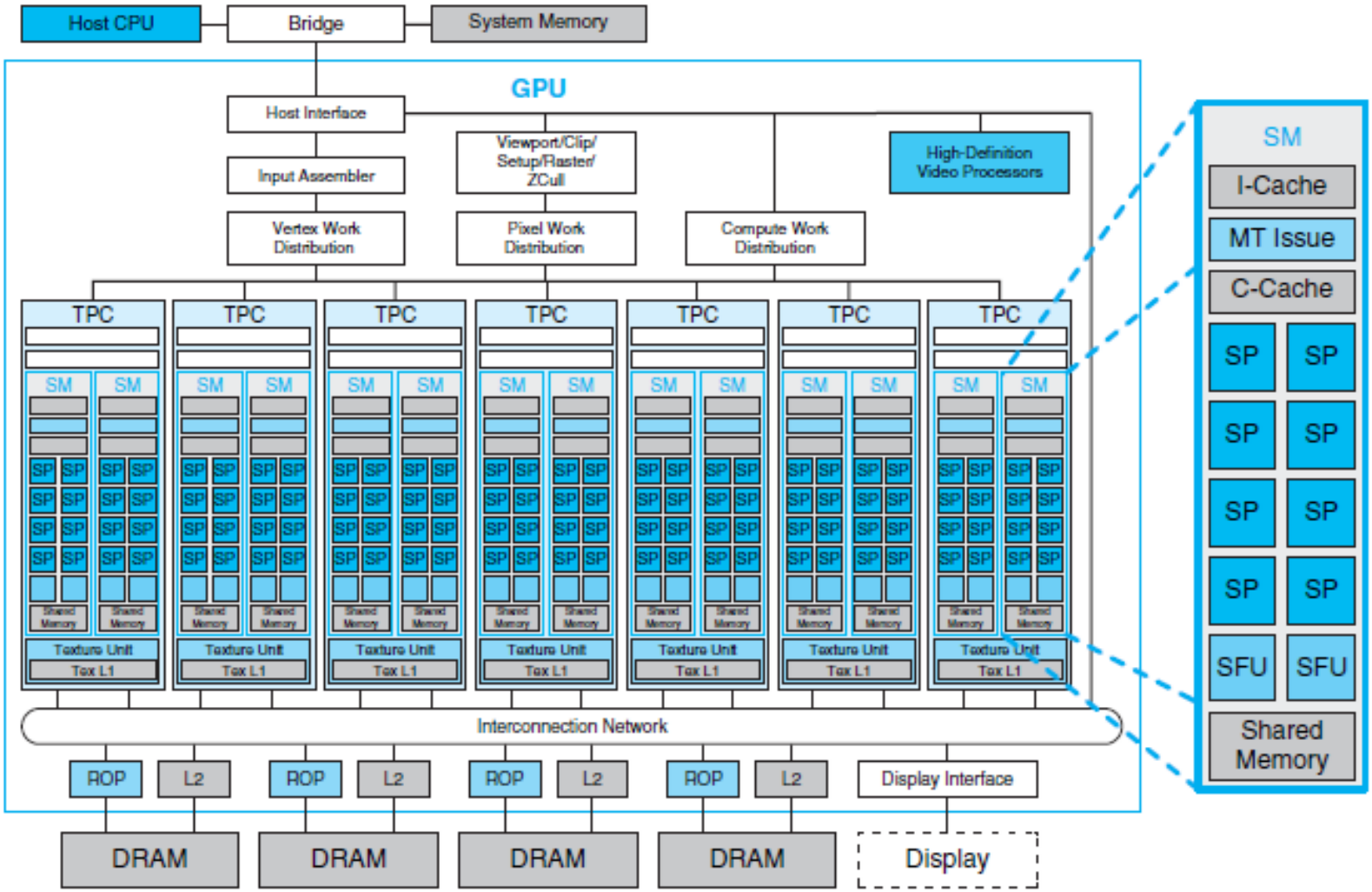
Unconditional Logistic Regression

- Propensity score models!!! (currently takes tens of hours with CV)
- Stratified models: separate indicator for each stratum

Conditional Logistic Regression

- Stratified models: condition out stratum, no dummy variables
- Ex: matched case control studies
- Many to many matching: can get computationally intensive

Graphics processing unit has thousands of threads that can execute the same piece of code (a.k.a. kernel) simultaneously.



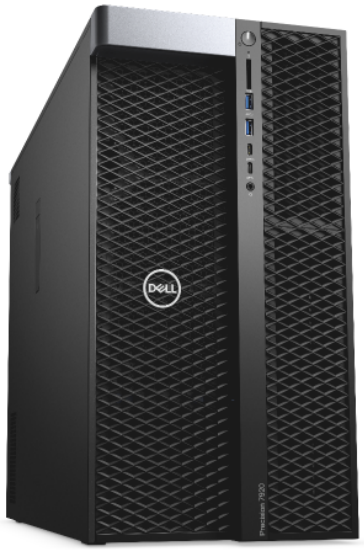
Why GPU?

Cloud Services

- Use cloud services every time running model
- Cost? $1 per hour
- Ease of use? hard

Why GPU?

Desktop



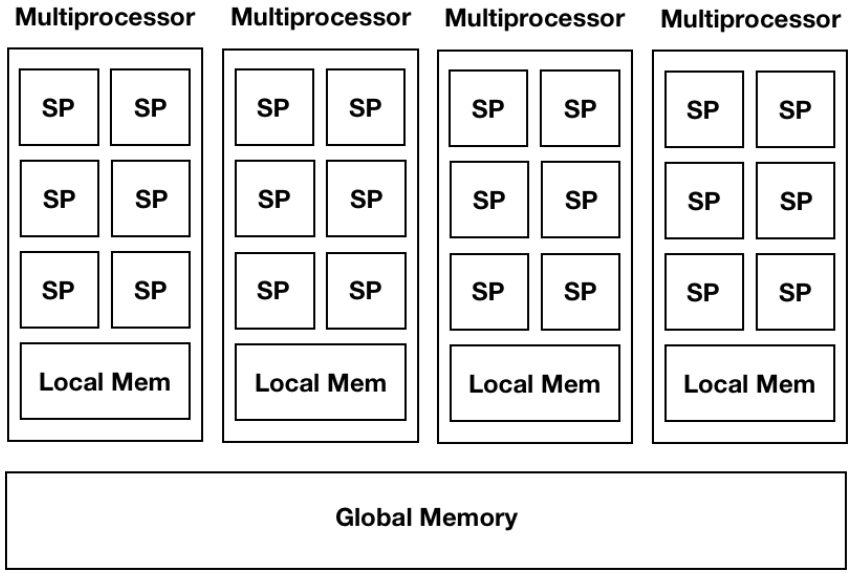
- Buy a monster desktop with 28 cores
- Cost? $5000
- Ease of use? medium

Why GPU?

GPU

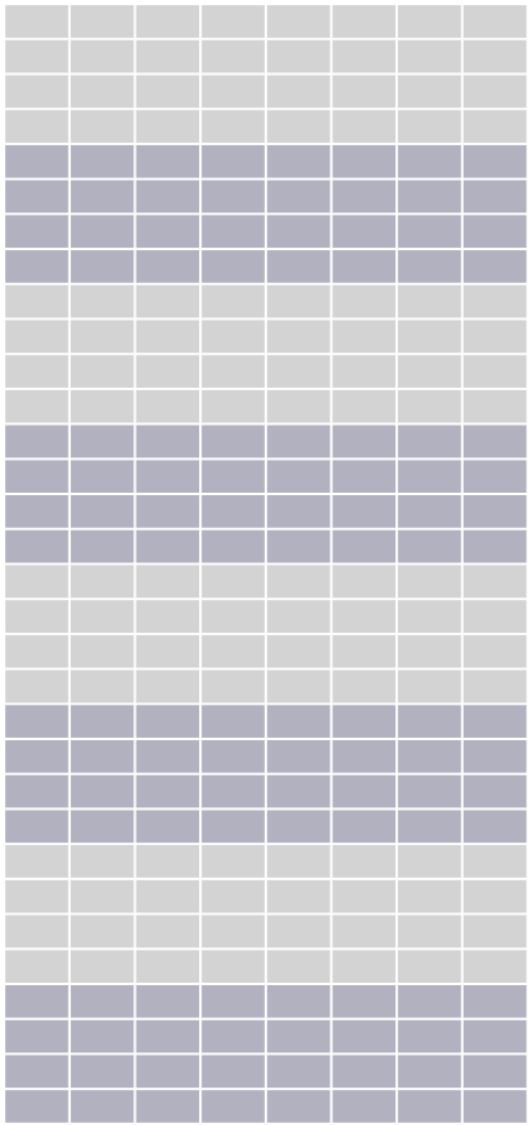
- Buy a GPU card
- Cost? $2000
- Ease of use? easy

How a GPU Works



- each Single Processor can run 32 threads simultaneously
- each Multiprocessor assigns “warps" of 32 threads to idle SPs
- warps belonging to same “thread block" run on a single multiprocessor and share local memory
- global memory access costs hundreds of clock cycles (primary performance bottleneck)
- NVIDIA Tesla K40 can run 2880 threads on 90 SPs across 15 multiprocessors

GPU Workgroup

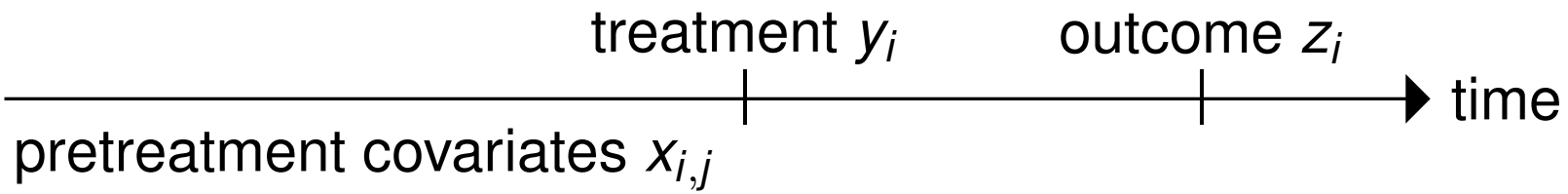


32x8 workgroup

- work group assigned to a multiprocessor
- work split into 8 warps of 32 threads
- each warp can be assigned to idle processing units
- all 32 threads in a warp run synchronously
- need to synchronize work across warps

Logistic Regression

- Fitting a propensity score with all pretreatment covariates



Log likelihood to maximize is:

$$l(\beta) = \sum_{i=1}^{n} y_i \boldsymbol{x}_i^T \beta - \log[1 + \exp(\boldsymbol{x}_i^T \beta)] + \lambda \sum_{j=2}^{p} |\beta_j|$$

Cyclic Coordinate Descent

- Updates one coordinate at a time with one-dimensional Newton steps

initialize initial search β;
while *not yet converged* **do**
  for $j \leftarrow 1$ **to** J **do**
    gradient $\leftarrow \frac{\partial}{\partial \beta_j} l(\beta) = \sum_{i=1}^{n} y_i x_{i,j} - \frac{x_{i,j} exp(\boldsymbol{x}_i^T \beta)}{1+exp(\boldsymbol{x}_i^T \beta)} + Sign(\beta_j)\lambda$;
    hessian $\leftarrow \frac{\partial^2}{\partial \beta_j^2} l(\beta) = -\sum_{i=1}^{n} \frac{x_{i,k}^2 exp(\boldsymbol{x}_i^T \beta)}{(1+exp(\boldsymbol{x}_i^T \beta))^2}$;
    delta $\leftarrow -$ gradient/ hessian;
    $\beta_j \leftarrow \beta_j$+delta;
    update $\boldsymbol{x}_i^T \beta$ and $exp(\boldsymbol{x}_i^T \beta)$ for subjects with $x_{i,j} \neq 0$;
  end
end

Algorithm 1: Cyclic coordinate descent

CCD Optimization

initialize initial search β;
while *not yet converged* **do**
  for $j \leftarrow 1$ **to** J **do**
    gradient $\leftarrow \frac{\partial}{\partial \beta_j} l(\beta) = \sum_{i=1}^{n} y_i x_{i,j} - \frac{x_{i,j} exp(\boldsymbol{x}_i^T \beta)}{1+exp(\boldsymbol{x}_i^T \beta)} + Sign(\beta_j)\lambda$;
    hessian $\leftarrow \frac{\partial^2}{\partial \beta_j^2} l(\beta) = -\sum_{i=1}^{n} \frac{x_{i,k}^2 exp(\boldsymbol{x}_i^T \beta)}{(1+exp(\boldsymbol{x}_i^T \beta))^2}$;
    delta $\leftarrow -$ gradient/ hessian;
    $\beta_j \leftarrow \beta_j$+delta;
    update $\boldsymbol{x}_i^T \beta$ and $exp(\boldsymbol{x}_i^T \beta)$ for subjects with $x_{i,j} \neq 0$;
  end
end

$\sum_{i=1}^{n} y_i x_{i,j}$ is constant, compute once

CCD Optimization

initialize initial search β;
while *not yet converged* **do**
  for $j \leftarrow 1$ **to** J **do**
    gradient $\leftarrow \frac{\partial}{\partial \beta_j} l(\beta) = \sum_{i=1}^{n} y_i x_{i,j} - \frac{x_{i,j} exp(\boldsymbol{x}_i^T \beta)}{1+exp(\boldsymbol{x}_i^T \beta)} + Sign(\beta_j)\lambda$;
    hessian $\leftarrow \frac{\partial^2}{\partial \beta_j^2} l(\beta) = -\sum_{i=1}^{n} \frac{x_{i,k}^2 exp(\boldsymbol{x}_i^T \beta)}{(1+exp(\boldsymbol{x}_i^T \beta))^2}$;
    delta $\leftarrow -$ gradient/ hessian;
    $\beta_j \leftarrow \beta_j$+delta;
    update $\boldsymbol{x}_i^T \beta$ and $exp(\boldsymbol{x}_i^T \beta)$ for subjects with $x_{i,j} \neq 0$;
  end
end

$\sum_{i=1}^{n} \frac{x_{i,j} exp(\boldsymbol{x}_i^T \beta)}{1+exp(\boldsymbol{x}_i^T \beta)}$ and $\sum_{i=1}^{n} \frac{x_{i,k}^2 exp(\boldsymbol{x}_i^T \beta)}{(1+exp(\boldsymbol{x}_i^T \beta))^2}$ can be distributed and reduced across multiple processes

CCD Optimization

initialize initial search β;
while *not yet converged* **do**
  for $j \leftarrow 1$ **to** J **do**
    gradient $\leftarrow \frac{\partial}{\partial \beta_j} l(\beta) = \sum_{i=1}^{n} y_i x_{i,j} - \frac{x_{i,j} exp(\boldsymbol{x}_i^T \beta)}{1+exp(\boldsymbol{x}_i^T \beta)} + Sign(\beta_j)\lambda$;
    hessian $\leftarrow \frac{\partial^2}{\partial \beta_j^2} l(\beta) = -\sum_{i=1}^{n} \frac{x_{i,k}^2 exp(\boldsymbol{x}_i^T \beta)}{(1+exp(\boldsymbol{x}_i^T \beta))^2}$;
    delta $\leftarrow -$ gradient/ hessian;
    $\beta_j \leftarrow \beta_j$+delta;
    update $\boldsymbol{x}_i^T \beta$ and $exp(\boldsymbol{x}_i^T \beta)$ for subjects with $x_{i,j} \neq 0$;
  end
end

updating $\boldsymbol{x}_i^T \beta$ and $exp(\boldsymbol{x}_i^T \beta)$ can be done across multiple independent processes

CCD on GPU

__local gradient[512], hessian[512];
task $\leftarrow$ id;
sum0, sum1 $\leftarrow$ 0;
while *task* $<N_j$ **do**
    k $\leftarrow$ K[task];
    xb $\leftarrow$ XB[k];
    sum0 $\leftarrow$ sum0 + $\frac{xb}{1+exb(xb)}$;
    sum1 $\leftarrow$ sum1 + $\frac{xb}{(1+exb(xb))^2}$;
    task $\leftarrow$ task + 512;
end
gradient[id] $\leftarrow$ sum0;
hessian[id] $\leftarrow$ sum1;
local parallel reduction to sum gradient and hessian;
if *id = 0* **then**
    delta $\leftarrow$ - gradient / hessian;
end
task $\leftarrow$ id;
while *task* $<N_j$ **do**
    k $\leftarrow$ K[task];
    XB[k] $\leftarrow$ XB[k] + delta;
    task $\leftarrow$ task + 512;
end

Performance Comparison

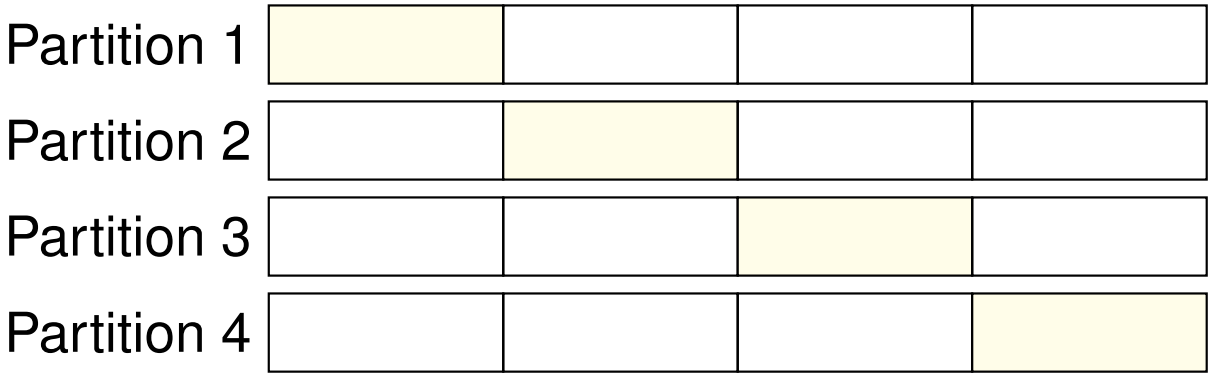
- Sparse n x p matrix (5% nonzero)
- 80% of β set to 0, rest are $Norm(0, 2)$
- Fit cyclops: model “lr", lasso without CV
- Compare Tesla K40 to my Macbook Pro (3.1GHz, 4 cores, 8GB)

n	p	iterations	GPU	CPU
5000	1000	95	10.6 s	18.5 s
50000	1000	114	14.1 s	296 s
5000	10000	156	146 s	214 s
50000	10000	409	466 s	8998 s (est)

- Launching a kernel is expensive
- To do: consolidate 3 steps into 1 kernel

Cross Validation

Search for optimal regularization hyperparameter is most expensive part of model fit



- Leave one fold out of each partition when training model
- Compute out-of-sample likelihood on the left out fold
- Running CCD on partitions is completely independent, can be parallelized

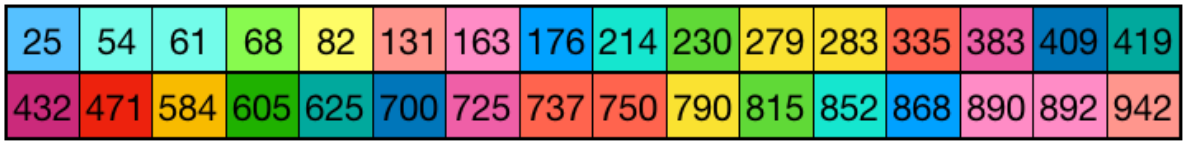
CCD on GPU

warp reads in 32 contiguous values from global K

k

25	54	61	68	82	131	163	176	214	230	279	283	335	383	409	419
432	471	584	605	625	700	725	737	750	790	815	852	868	890	892	942

k



- Our problems are sparse, so k are non-contiguous
- Here, warp will request 28 blocks from global memory
- Will need to wait for 500 x 28 = 14000 clock cycles
- Memory access is not coalesced

Cross Validation on GPU

- Naive way to run different partitions: run each partition in separate work group or kernel
- Each partition accesses the same indices of a different XB vector
- 28 x 4 = 112 global memory accesses

Cross Validation on GPU

Partition:	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
	25	25	25	25	54	54	54	54	61	61	61	61	68	68	68	68
	82	82	82	82	131	131	131	131	163	163	163	163	176	176	176	176
	214	214	214	214	230	230	230	230	279	279	279	279	283	283	283	283
	335	335	335	335	383	383	383	383	409	409	409	409	419	419	419	419
	432	432	432	432	471	471	471	471	584	584	584	584	605	605	605	605
	625	625	625	625	700	700	700	700	725	725	725	725	737	737	737	737
	750	750	750	750	790	790	790	790	815	815	815	815	852	852	852	852
	868	868	868	868	890	890	890	890	892	892	892	892	942	942	942	942

- Improved way to run different partitions: lay out global memory XB so partitions access contiguous memory
- Each partition accesses a different XB value of the same index, but are next to each other
- 29 global memory accesses

Performance Comparison

- Sparse n x p matrix (5% nonzero)
- 80% of β set to 0, rest are $Norm(0, 2)$
- Fit cyclops: model “lr", lasso with cross validation
- Compare Tesla K40 to my Macbook Pro (3.1GHz, 4 cores, 8GB)

n	p	CV	iterations	GPU	CPU
1000	200	10 x 1	93	1.2 s	5.5 s
1000	200	10 x 10	101	1.4 s	47.5 s
1000	1000	10 x 1	298	14.6 s	81.3 s
1000	1000	10 x 10	409	14.4 s	738 s
10000	5000	10 x 1	928	202 s	9579 s (est)
10000	5000	10 x 10	1270	181 s	36.3 h (est)

- GPU performance dominated by global memory access

Performance Comparison

- Warfarin vs dabigatran dataset
- Call createPs from CohortMethod
- Fit cyclops: model “lr", lasso with cross validation
- Compare Tesla K40 to iMac (3.9GHz, 8 cores, 16GB)

n	p	CV	iterations	GPU	CPU
74624	3626	10 x 10	101	6.48 m	46.1 m
74624	24815	10 x 1	760	1.39 h	
74624	24815	10 x 10	4663	3.14 h	12.5 h

- GPU several times faster for 10x10 CV
- Shows the parallelization limits of LR
- Might not be better for 10x1, but could also be running multiple lambda

Unconditional Logistic Regression (ULR)

Introduces [linear] stratum level effects:

- K total strata, $\{S_k\}$ set of people in stratum k
- Introduce new covariates $x_{i,p+1}, x_{i,p+2}, \cdots x_{i,p+K}$
- and coefficients $\beta_{p+1}, \beta_{p+2}, \cdots \beta_{p+K}$
- New coefficients are “nuisance parameters"

Log-likelihood:

$$l(\beta) = \sum_{k=1}^{K} \sum_{i \in \{S_k\}} y_i \boldsymbol{x}_i^T \beta - \log[1 + \exp(\boldsymbol{x}_i^T \beta)]$$

where $\boldsymbol{x}_i$ and β are expanded $p + K$ length vectors

Conditional Logistic Regression (CLR)

Condition on number of outcomes in each stratum

- K total strata, m_k cases out of n_k people in stratum k
- $\{p_k\}$ set of cases in stratum k
- Condition on that m_k out of n_k were cases
- Likelihood that the observed cases were those m_k cases

Likelihood:

$$l(\beta) = \prod_{k=1}^{K} \frac{\prod_{i \in \{p_k\}} \boldsymbol{x}_i^T \beta}{\sum_{\{p'\} \in R_k} \prod_{j \in \{p'\}} \exp(\boldsymbol{x}_j^T \beta)}$$

where R_k are the set of all $\binom{n_k}{m_k}$ combinations of having m_k cases out of n_k in stratum k

CLR vs ULR

Pros:

- Avoids estimating nuisance parameters $\rightarrow$ potentially less bias
- Preferred for case control studies (lots of strata)

Cons:

- Much more work than regular logistic regression (1 order more)
- Fast approximations (Breslow, Efron) inaccurate
- Large strata cause numerical overflow

CLR Recursion

$$\frac{\prod_{i\in\{p\}} \boldsymbol{x}_i^T\boldsymbol{\beta}}{\sum_{\{p'\}\in R}\prod_{j\in\{p'\}}\exp(\boldsymbol{x}_j^T\beta)} = \frac{\prod_{i=1}^{m} U_i}{\sum_{\binom{n}{m}}\prod_{j=1}^{m} U_j} = \frac{\prod_{i\in\{p\}} U_i}{B(m,n)}$$

- Combinatoric term $B(m,n)$ sums over $\binom{n}{m}$ terms
- Can be simplified using relation $\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$

$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$

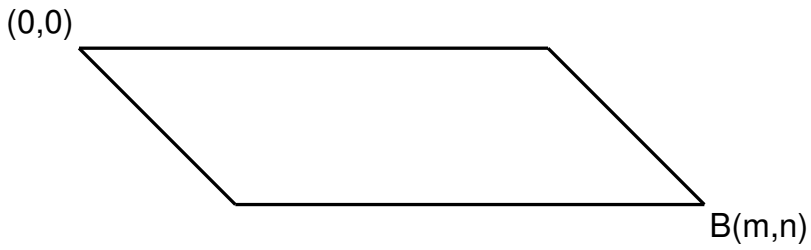
For CCD, first $B'(m,n)$ and second $B''(m,n)$ derivatives in some coordinate β_r are:

$$\begin{aligned} B'(m,n) &= B'(m,n-1) + U_n B'(m-1,n-1) + x_{nr} U_n B(m-1,n-1) \\ B''(m,n) &= B''(m,n-1) + U_n B''(m-1,n-1) + x_{nr} U_n B(m-1,n-1) \\ &\quad + 2x_{nk} U_n B'(m-1,n-1) \end{aligned}$$

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

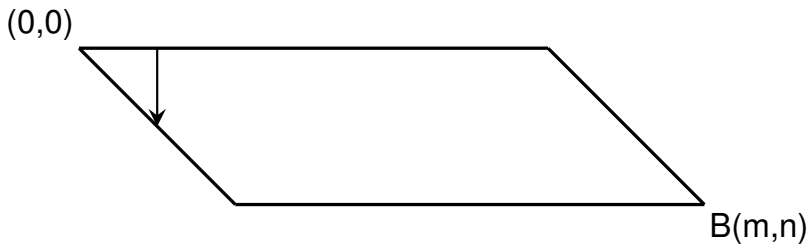
Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

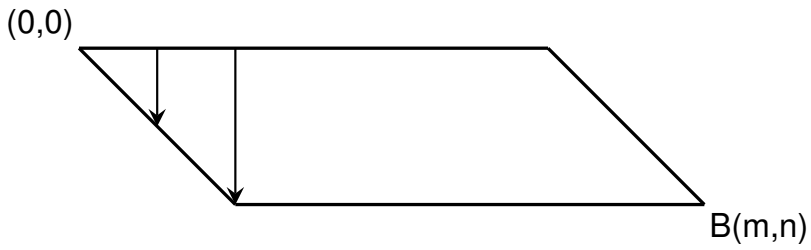
Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

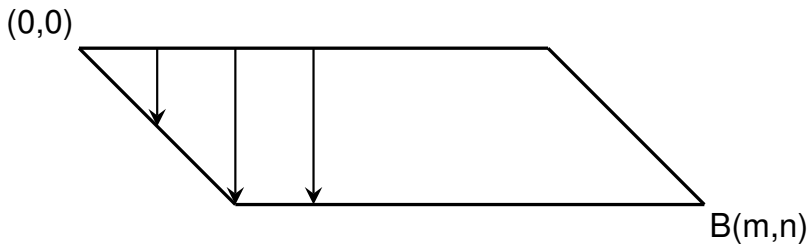
Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

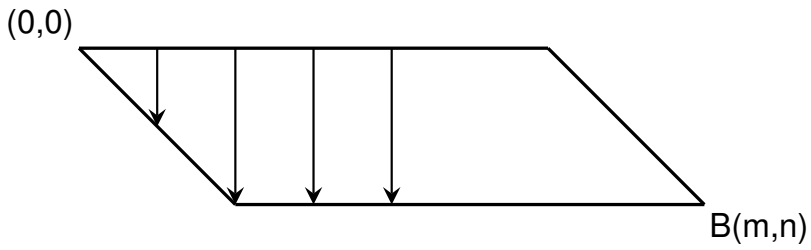
Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

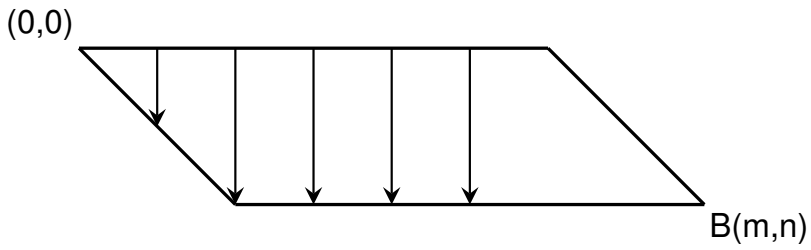
Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

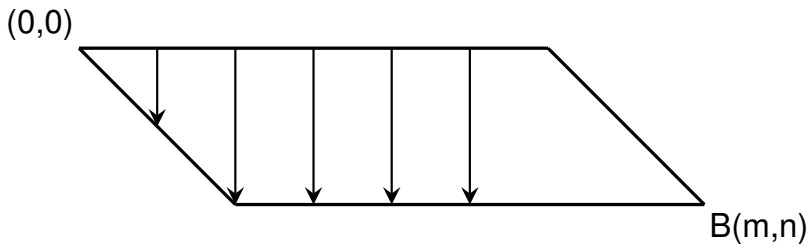
Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Parallelization Opportunities

Table 1. *Recursive generation for* $B(m, n)$ *for* $m = 2$, $n = 3$

m \ n	0	1	2	3
0	1	1	—	—
1	0	U_1	$U_1 + U_2$	—
2	0	0	$U_1 U_2$	$U_1 U_2 + U_1 U_3 + U_2 U_3$



$$B(m,n) = B(m,n-1) + U_n B(m-1,n-1)$$
$$B'(m,n) = B'(m,n-1) + U_n B'(m-1,n-1) + x_{nk} U_n B(m-1,n-1)$$
$$B''(m,n) = B''(m,n-1) + U_n B''(m-1,n-1) + x_{nk} U_n B(m-1,n-1) + 2x_{nk} U_n B'(m-1,n-1)$$

Note:

- Read x_{nk} and U_n once, update entire column (parallelizable)
- More parallelization opportunity than LR

Cyclic Coordinate Descent

- Updates one coordinate at a time with one-dimensional Newton steps

initialize initial search β;
while *not yet converged* **do**
  for $j \leftarrow 1$ **to** J **do**
    gradient $\leftarrow \frac{\partial}{\partial \beta_j} l(\beta) = \sum_{i=1}^{n} y_i x_{i,j} + \sum_{k=1}^{K} \frac{B_k'}{B_k}$;
    hessian $\leftarrow \frac{\partial^2}{\partial \beta_j^2} l(\beta) = \sum_{k=1}^{K} \frac{B_k''}{B_k} - (\frac{B_k'}{B_k})^2$;
    delta $\leftarrow -$ gradient/ hessian;
    $\beta_j \leftarrow \beta_j$+delta;
    update $\boldsymbol{x}_i^T \beta$ and $exp(\boldsymbol{x}_i^T \beta)$ for subjects with $x_{i,j} \neq 0$;
  end
end

Algorithm 2: Cyclic coordinate descent

CLR on GPU

```
__local B0[2][128], B1[2][128], B2[2][128];
setup initial values;
int c = 0;
for col <stratumSize do
    x ← X[col];
    U ← EXB[col];
    B0[1-c][id] = B0[c][id] + U*B0[c][id-1];
    B1[1-c][id] = B1[c][id] + U*B1[c][id-1] + x*U*B0[c][id-1];
    B2[1-c][id] = B2[c][id] + U*B2[c][id-1] + x*U*B0[c][id-1] + 2*x*U*B1[c][id-1];
    c = 1 - c;
end
if id = 0 then
    gradient[stratum] ← B0[c][cases];
    hessian[stratum] ← B0[c][cases];
end
parallel reduction to sum over strata to get delta;
task ← id;
while task <N_j do
    k ← K[task];
    XB[k] ← XB[k] + delta;
    task ← task + 512;
end
```

Numerical Challenges

Problem: numerical overflow with large strata

- Using Stirling's approximation, $\log_2 \binom{n}{n/2} = n + O(\log_2 n)$
- If $U_i \approx 1$, $B(n, n/2) \approx 2^n$
- Double precision floating point goes up to 2^{1023}
- So for strata and cases in the high hundreds or more, overflow arises
- Not a problem for any small to moderate sized strata

Solutions:

- Rescale columns as needed ($x = x/10^{200}$) $\rightarrow$ inaccurate, falls apart for large n
- long double format,15 bits for exponent ($2^{2^{14}-1}$) $\rightarrow$ expensive, not for GPU
- Work in log domain $\rightarrow$ even more expensive

Numerical Challenges

Problem: numerical overflow with large strata

- Using Stirling's approximation, $\log_2 \binom{n}{n/2} = n + O(\log_2 n)$
- If $U_i \approx 1$, $B(n, n/2) \approx 2^n$
- Double precision floating point goes up to 2^{1023}
- So for strata and cases in the high hundreds or more, overflow arises
- Not a problem for any small to moderate sized strata

Solutions:

- Rescale columns as needed ($x = x/10^{200}$) $\rightarrow$ inaccurate, falls apart for large n
- long double format,15 bits for exponent ($2^{2^{14}-1}$) $\rightarrow$ expensive, not for GPU
- Work in log domain $\rightarrow$ even more expensive

Performance Comparison

- Dense n x p matrix (5% nonzero) with k strata
- β drawn from $Norm(0, 2)$
- Linear stratum effects drawn from $Norm(0, 0.25)$
- No regularization
- Compare Tesla K40 to my Macbook Pro (3.1GHz, 4 cores, 8GB)

		clogit		glm ULR		Cyclops CPU		Cyclops GPU	
n / p / k	cases	Time	Bias	Time	Bias	Time	Bias	Time	Bias
2k/100/100	8.3/20	11.5	0.411	0.48	0.570	12.6	0.411	6.2	0.411
20k/100/1k	13.3/20	313	0.109	334	0.217	112	0.109	10.9 s	0.109

Rest to do

- Compare CV CLR on GPU to clogit.L1 package
- Implement Breslow approx for GPU
- More comparisons