Github url: [git@github.com:andrew841018/disksim\_github.git](mailto:git@github.com:andrew841018/disksim_github.git)

Branch:

* Original….額，我也不知道是幹嘛用的
* Before-simulation-finish-data-collect: 主要是在很早以前收集完資料的版本，基本上可以忽略
* Page-striping: 全部以page striping方式做擺放，但程式本身尚未完完善，有一些問題未解決
* Build-model-after-simulation: 用來收集資料用的，裡面的【disksim\_2021(all buffer)】(至於ignore的用途，讀完RC-RNN這篇論文就會懂了)可用來收集資料，也就是【sector num-physical block num-benefit-sector count.txt】，但實際上，最後完成整個程式後，用完整版的程式跑，所產生出來的資料，會讓hit ratio更高，所以…基本上這個branch到後期是沒用的。
* Rewrite-disksim\_2021-base-on-simulation-info: 主要的程式，分成AI、AI+Hint(use multiply method)…..以下會詳記解釋每個資料夾的用途。
  + AI+Hint(use multiply method): 將AI與學長的Hint queue做結合，的程式，在這裡會去讀duration.txt，目的是抓取每個block對應的duration label(也就是透過LSTM預測的結果)
  + Dynamic-striping-AI: 其實就是單純用AI預測結果來選victim block的程式，這裡同樣也會去讀duration label
  + All version of ssd.c: 主要是儲存不同時間點的AI+Hint ssd.c程式，每個程式的實作方式略微不同，但都可以執行。
  + BLAS(can run): 由於從NAS載下來的BLAS是沒有Page cache的，因此這裡加上了Page cache，讓進入write buffer的reuqest與學長的程式是一致的，否則這兩個程式的write buffer即便是用同個trace，所接收到的request也會完全不同。
  + LPB(can run):同blas，也是沒有page cache，我特別加上了page cache的程式。
  + Collected data(from disksim): 檔名別理他，實際上裡面大部分都是AI相關資訊
    - Duration value: 從build-model-after-simulation所產生的duation value，格式為【block number,duration value】，但用實際去跑的程式來產生的話，實驗結果應該會更好，畢竟兩者架構不太一樣。
    - Benefit: 產生【sector num-physical block num-benefit-sector count.txt】，其餘同上。
    - AI input feature: 產生【info(trace\_name).txt】(送進AI model的input feature)，trace\_name是指trace的名稱，如: iozone,Financial2forssd….，其餘同上。
      * Generate by current code: 表示是用當下程式所產生的input feature而非build-model-after-simulation中的ssd.c，代表實驗結果會更好。
    - All buffer-duration\_label: 在python中將duration value做轉換，當成model answer，訓練完model，並用訓練好的model預測前面產生的AI input feature所得到的結果(也就是duration label，格式是: block number, duration label)。
      * Duration label com from current code: 表示是用當下程式所產生的input feature拿去給AI model訓練，而非來自build-model-after-simulation中的ssd.c，代表實驗結果會更好。
  + Disksim\_new\_method: 學長程式，在NAS可以找到一模一樣的資料夾，然後你就知道是哪個學長的程式了。
  + Disksim\_with\_command: 由學長程式修改的，所有我所理解的部分、後來看懂的部分，詳細註解都會寫在裡面的ssd.c
  + For\_testing(back up): 唯一不同的只有AI+Hint，裡面的ssd.c與all version of ssd.c中的【ssd(kick 8 page with How and 3 queue).c】相同。
    - 簡單來說，他先將64個候選victim block標記起來，若其中某個block被overwrite，就解除標記，然後當標記的數量少於64個，就又會挑選victim block標記，直到標記數量=64。
    - 踢資料的時候，以8 page為單位去踢，直到write buffer有空的位置可以讓新進的request寫入，而踢的page不會是同一個block，而會挑出在每個channel中，loading最輕的那個plane，對應的block，挑一個標記過(mark)的page踢掉。
    - 因此實際上在8個block中，各挑一個page做踢除，此作法可減少GC，但hit ratio也跟著下降就是了…(也是學長程式的做法)。
  + Original BLAS+LPB(it need some modify): 檔名應該很清楚了，就是沒有Page cache的LPB & BLAS
  + Rest: 包含三種不同結合Hint的方式、以block striping做擺放、將benefit<min benefit in write buffer的request ignore、在學長程式上實作oracle的程式(都有bug…)
  + Testing(run\_all\_trace): 檔名來自於，我原本打算切割trace(因為後面會有bug…)，但後來解完bug了，檔名擺著也沒改，基本上AI+Hint、dynamic-striping-AI是與根目錄上看到的是一樣的。後面我用來平行跑很方便，畢竟相同程式兩分，就可以一次跑兩個(其實for testing當中的dynamic-striping-AI也一樣，所以嚴格說來dynamic-striping-AI可以一次跑三個)
    - By the way,目前實測，只要你的程式沒問題，虛擬機一次可以開12個程式一起跑(12個以上會開始變很慢….)
  + Trace
    - 產生trace相關資訊: 裡面有產生write ratio、sequential write ratio的程式，可以輸入input trace & output file name，然後程式就會根據你給的output file name新創一個output file，並將資訊寫入。
  + Txt file: 環境設定、執行disksim所要輸入的命令(by the way，你要先make成功)
  + 論文相關: 總之裡面不只有論文，有很多可以用到的解釋文件，就，自行觀看…
  + BLAS.zip、LPB.zip: 對應上面【original BLAS+LPB(it need some modify)】，兩個是同樣的檔案
  + Test.c: 測試用的小程式
  + Git instruction: 如果有用git，可以省你很多時間，不過關於大檔案無法上傳這點，我認為還是LFS方案買下去，一個月300$可以省掉你非常非常非常多的時間。(處理git 大檔案上傳造成的錯誤真的很煩…)
  + Makefile.txt: 裡面的makefile已被修改過，每次make會自動刪除syssim\_driver.o & ssd.o，這樣修改的程式會反映在結果上(否則你可能修改一大堆，結果都沒變)。這個檔案可用來取代disksim目錄下的makefile