- Mapper で生成する key は重複していても良いのか？（p.8）
  - 大丈夫。Key ごとに分類されて Reducer に渡される
- Mapper/Reducer の個々の処理粒度はどのくらいの時間を目安にすればよいのか？分単位くらいと考えていれば良いのか？（p.10）
  - 目安の設定は難しいが、分単位であれば問題はないだろう。実際には Mapper/Reducer 実行のオーバーヘッドも考慮した方が良い。オーバーヘッドは処理系依存。
- KMR プログラム実行中にマシンがダウンしてしまったときは全部ダウンしてしまうのか？（p.17）
  - 現状の実装ではプログラム全体がダウンする。ただ、実行状態の保存・再開機能（Checkpoint/Restart）を実装しているので、中断時の状態から再開することは可能
- C99 に準拠しなければならない理由は何なのか？（p.17）
  - KMR 本体が C99 準拠で実装されているだけであり、ユーザプログラムは準拠の必要はない。実際 Fortran、C++でもプログラムは実装可能。
- 乱数はどのように生成するのか？並列乱数を使用しているのか？（p.30）
  - サンプルでは乱数は rand()で生成している。KMR のライブラリを用いたプログラミングを行えば、並列乱数生成ライブラリと組み合わせてプログラム実装できる。ただ、KMRRUN を使う場合はそのようなことはできない。
- Mapper が一つ目を呼び出したときは 000（ファイル名）でその次に呼び出すのが 001（ファイル名）なのか？（p.33）
  - 順番通りに起動されるが、実際には確保した（動的）プロセス数分同時に並列に起動される。
- プロセッサ数 2 個で KMRRUN を実行するとどうなるのか？（p.34）
  - KMRRUN の実行並列数と KV Generator の実行並列数は等しい。このため、KV Generator も 2 プロセスで実行される。Mapper の出力サイズが大きい場合、KV Generator の実行コストも上がることが考えられるため、このような場合には KMRRUN の並列数を上げた方がよいことがある。Mapper しか実行しない場合、KV Generator の実行コストが小さい場合には KMRRUN は 1 プロセスで実行しても問題無い。
- 実行時の並列数を意識して指定しないと遊休リソースが生じてしまうが、

どう注意すればよい？（p.58）

- 可能ならばアルゴリズムレベルで修正し、Mapper の入力数（入力ファイル）と Reducer の入力数（Mapper 出力 Key 種類）を合わせるようにする。

- ゲノム解析で、IO を行う Merge をオンメモリ通信の Shuffule に変更したことだけが有意差？（p.65）
  - そうである。実際に他の処理は全く一緒である。
- レプリカ交換数が増えれば KMR のほうが有利か？（p.70）
  - 有意差はないと思われる。
- Reducer を実装できるのが売りだと思うのだがどうやって実装するのか？自由に書けるのでどこかの処理に特化して速いというということがあるのか？
  - 同一 Key を持つ複数の KV を集計する処理として実装する必要がある。特定の処理に対して最適化は行っていない。
- 京コンピュータでしか使えないのか？
  - Fujitsu FX10 や任意の Linux クラスタで利用可能
- MR-MPI というのは別のソフトなのか？機能の違いは何か？MR-MPI もノードの中で MPI を使えば速くなるのではないか？
  - 別ソフトである。MR-MPI ではノードに並列はされないため、全コアを活用するには､全コア数分の MPI プロセスを立ち上げる必要があり、数万コア規模になるとスケールしない。一方、独自のページングの仕組みで KV を管理しており、ノードのメモリ量を超えた KV の処理が可能。KMR の場合、ノードメモリが枯渇すると強制終了となる。