並列プログラミングコンテスト 非数値部門　成果報告

筑波大学　鈴木克典, 米元大我

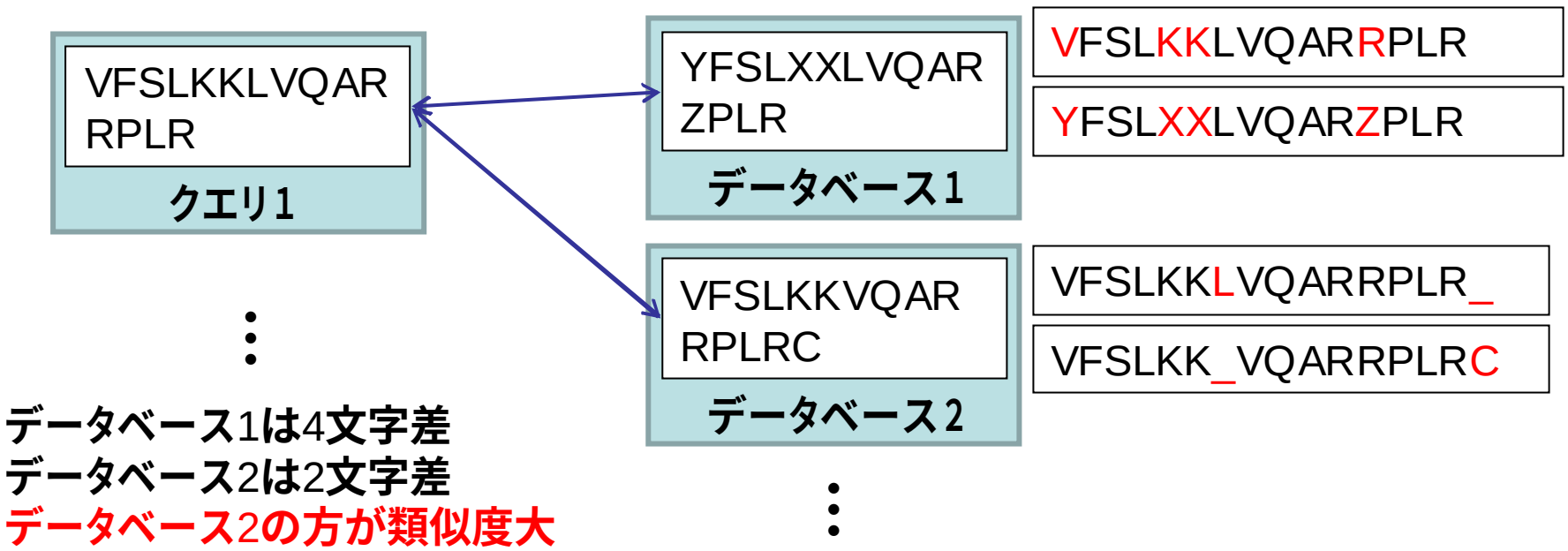
2009/5/27

HA8000クラスタシステム ~T2K-Tokyo~

- 初めてのバッチインタフェース
- 標準の日立コンパイラ、gcc、インテルコンパイラではコンパイルエラー
  - PGIコンパイラで解決（？？）
- デバッグの難しさ
- T2K-Tsukubaと比べて
  - ○ homeディレクトリの扱いやすさ
  - ✕ ‘ls’コマンドが返ってこない（？）

非数値部門 課題の概要 (1)

- 相同性検索プログラムの並列化
  - 与えられたクエリー配列群に対し，それぞれ既知のデータベース配列群から最も類似度の高いものを検索
  - クエリ配列，データベース配列は文字列で表現される
  - サンプルのSmith-Waterman プログラムを並列化



非数値部門 課題の概要（2）

- サンプルプログラムの計算量
  - アルゴリズムの計算量はクエリの文字数q，データベースの文字数dの積 - O(qd)
  - 全てのクエリーに対し，Smith-Waterman アルゴリズムにより全データベースを探索
    - クエリ配列の数Q，データベース配列の数Dのとき Q*D 回の探索
  - 総計算量＝（Q×D）×O(qd)

各問題セットの傾向

セット	q1		q2		q3		q4		q5	
長さ	Q	DB	Q	DB	Q	DB	Q	DB	Q	DB
0	20,566	201	2,099	2,061	214	20,550	197	0	42	0
500	3,738	34	328	362	32	3,718	45	0	6	0
1,000	335	6	33	37	1	336	2	0	2	0
1,500	287	9	35	33	1	293	5	0	0	0
2,000	20	0	1	1	0	26	0	0	0	0
2,500	22	0	2	3	0	32	0	0	0	0
3,000	19	0	2	1	1	26	1	0	0	0
3,500	13	0	0	2	1	19	0	0	0	0
1万-7万	0	0	0	0	0	0	0	250	0	0
40万-180万	0	0	0	0	0	0	0	0	0	50
total	25,000	250	2,500	2,500	250	25,000	250	250	50	50

Q:query DB:database

問題分析

- サンプルプログラムプロファイル例

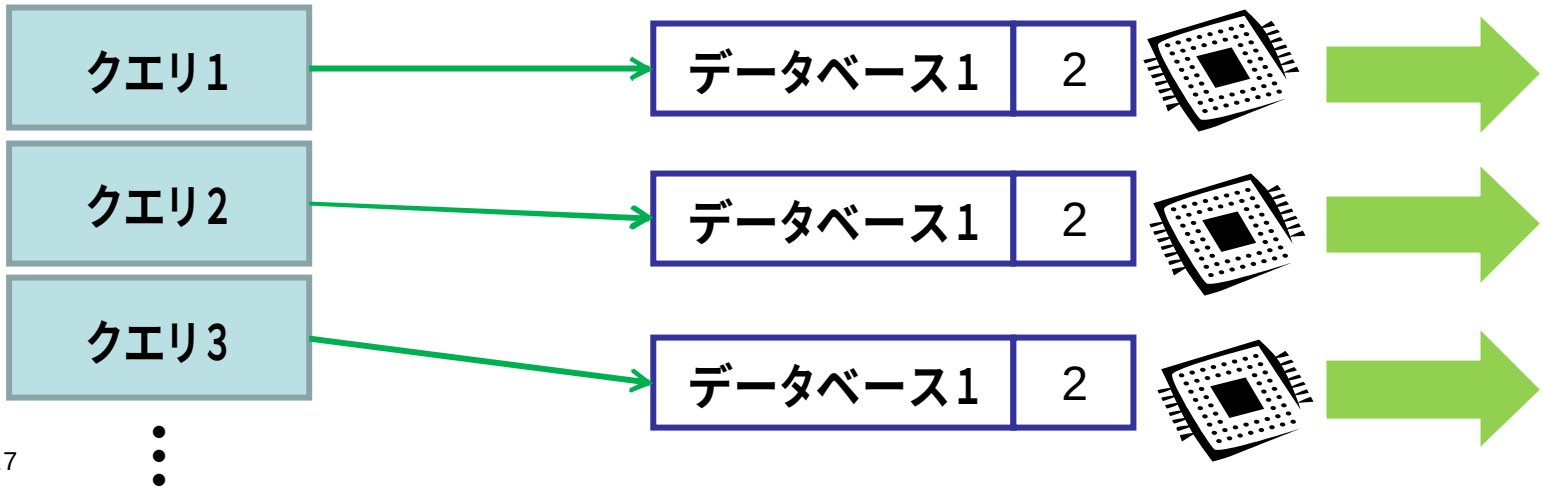
```
ksuzuki@les00:sample_profile% gprof sw-sample gmon.out
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 97.03   1868.51  1868.51  1440000     1.30     1.30  get_smith_waterman_score
  3.06   1927.47    58.95  1440000     0.04     0.04  show_alignment
  0.00   1927.50     0.03                             main
  0.00   1927.52     0.02        2    10.01    10.01  load_sequence_set
  0.00   1927.52     0.00    12120     0.00     0.00  set_seq_name
  0.00   1927.52     0.00        1     0.00     0.00  load_score_matrix
```

- Smith-Waterman**による検索自体の実行時間は比較的小さいが, 呼び出し回数が非常に多い**
- **様々なパターンの問題セットがあるがほぼ同じ傾向**
  - **クエリ, データベース文字数が小さく, クエリ集合, データベース集合の数が多い**

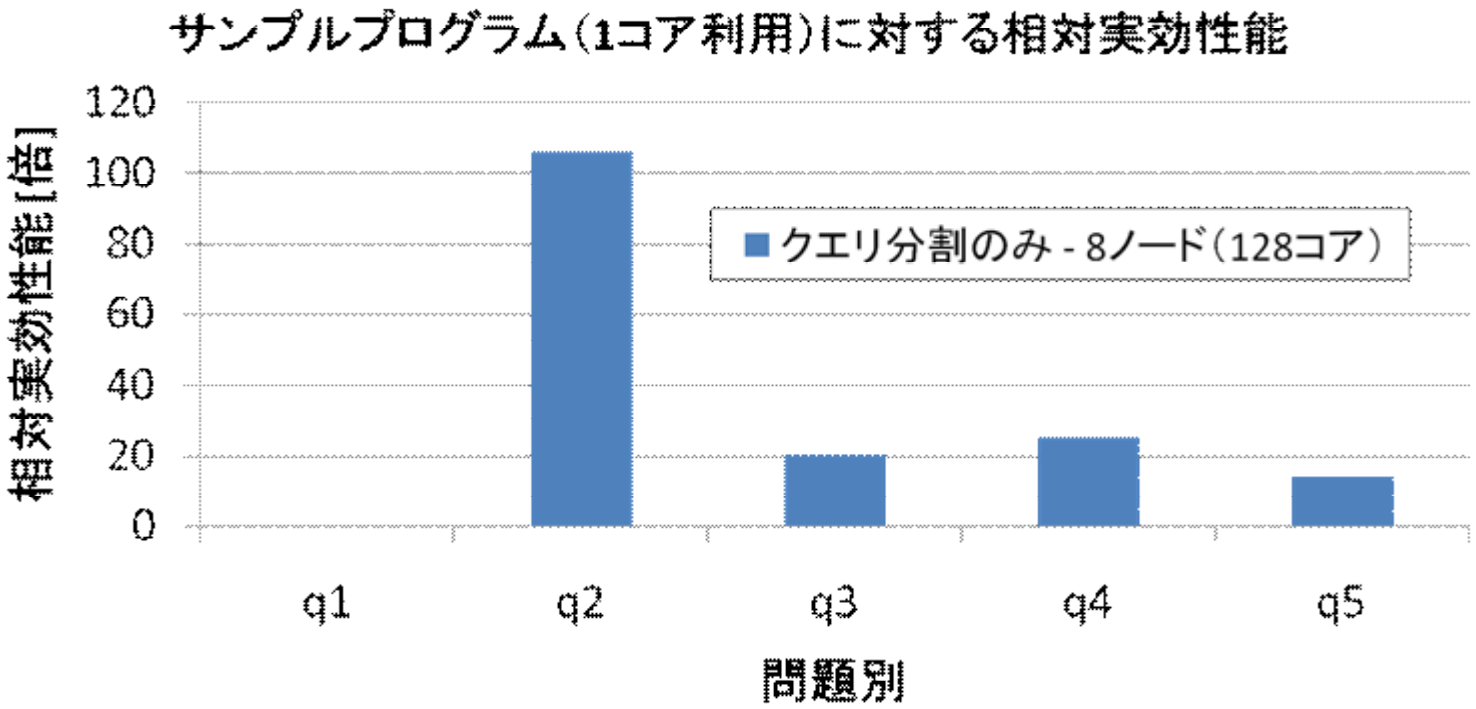
並列化の方針

- アルゴリズム自体はサンプルのまま.
- 呼び出し回数を減らすためにタスク (クエリ×データベース) を各コアに分散させる

- クエリ配列の分割
  - Smith-Waterman アルゴリズムにはすべてのデータベースが渡されるため, すべてのクエリは独立に検索可能



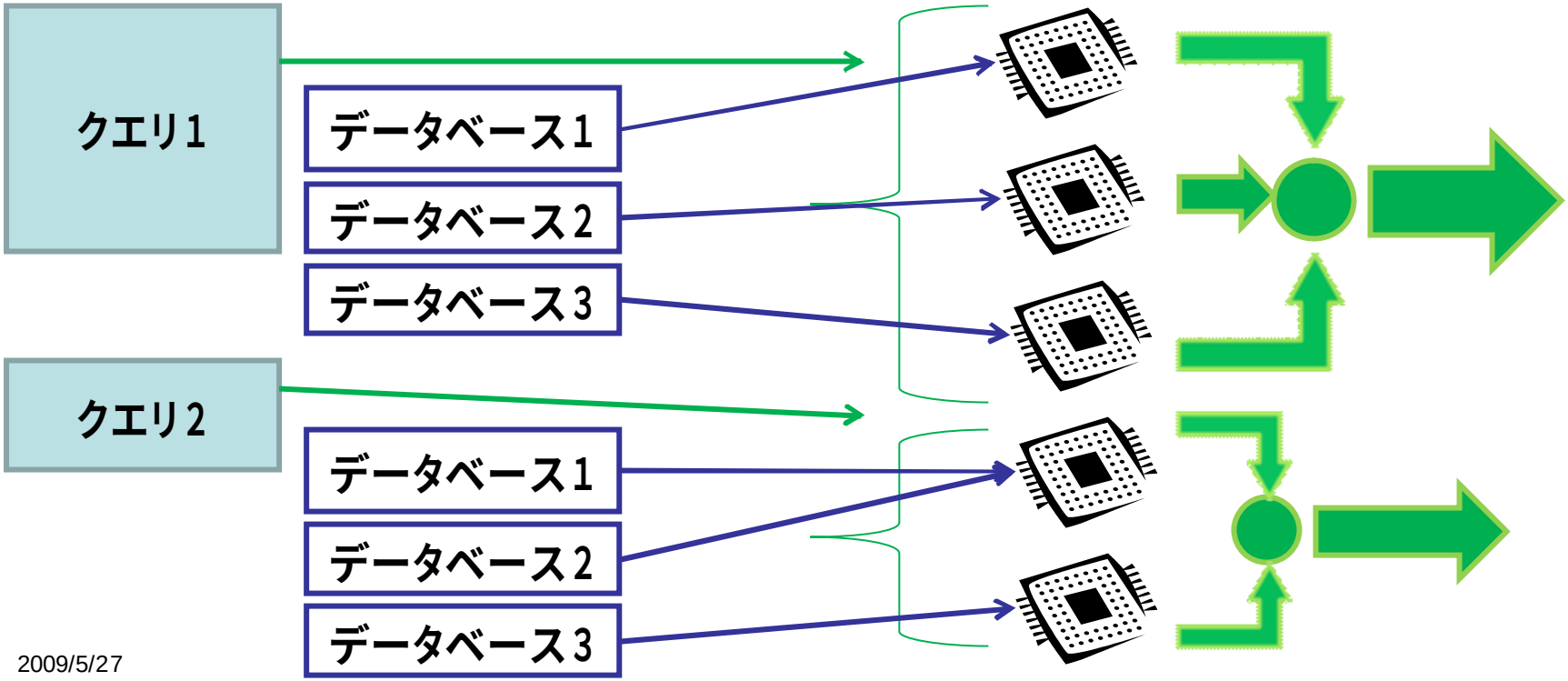
予選を終えて



- クエリ数が少なくてデータベースが大きいときは性能が出ていない
- クエリ分割の限界
  - 改良が必要

クエリ分割＋データベース分割並列化概要

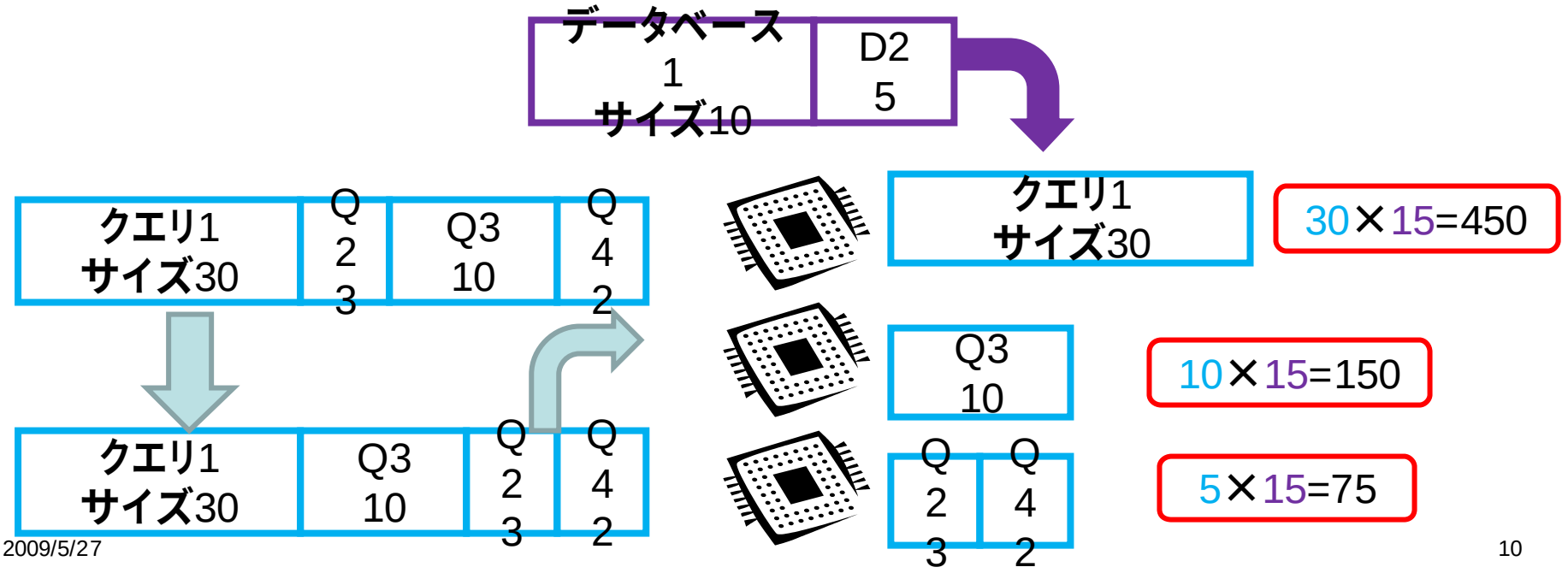
- クエリ配列分割とデータベース配列分割の組み合わせ
  - 基本的にクエリ配列分割でクエリの大きさに応じてデータベースも分割する
  - データベース配列分割の同期オーバーヘッドを軽減
  - すべてのコアの計算量を平均化する



タスク-コア割り当てアルゴリズム (1)

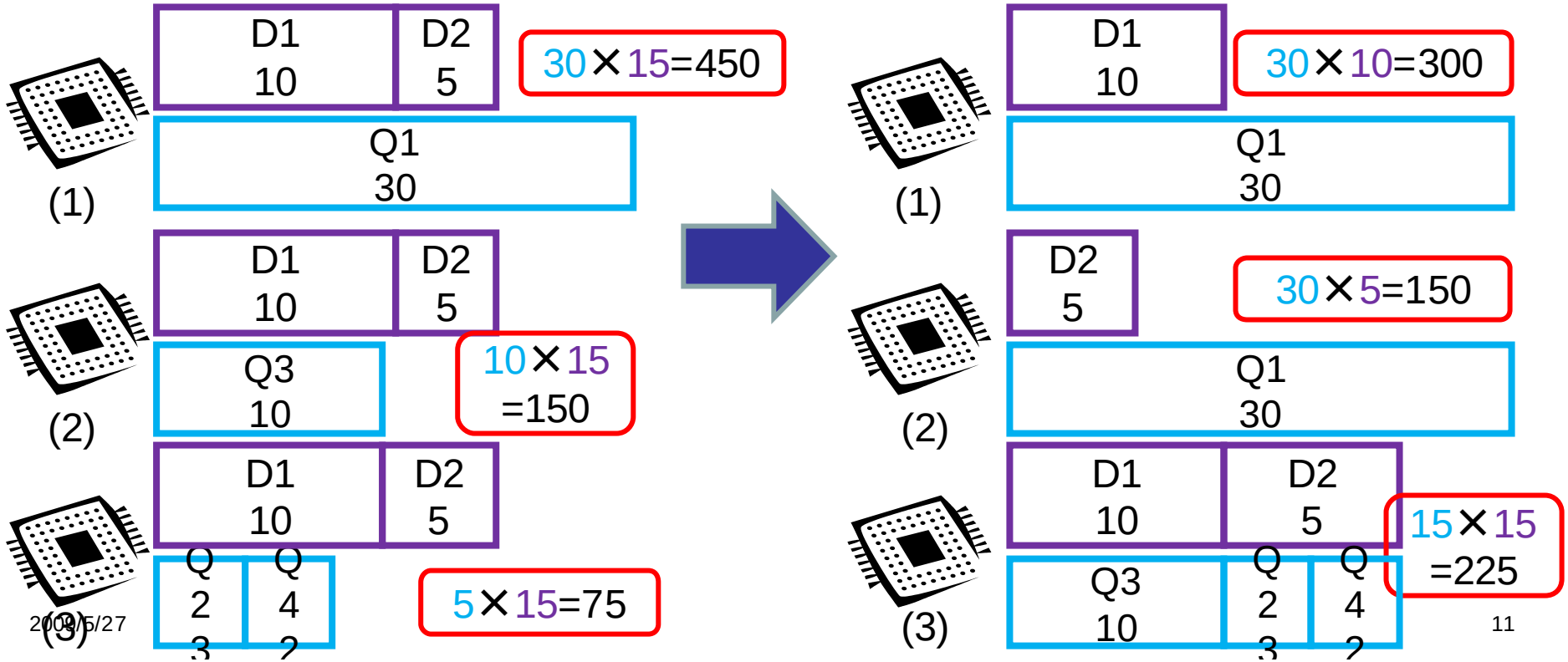
- クエリ (データベース) はそれぞれ独立・依存性なし
  - 簡単な逐次リストスケジューリングを使用
  - 基準はクエリ (データベース) の文字数

1. クエリをリストスケジューリングによりすべてのコアに割り当て
   - データベースはすべてのコアが持つと仮定



タスク-コア割り当てアルゴリズム (2)

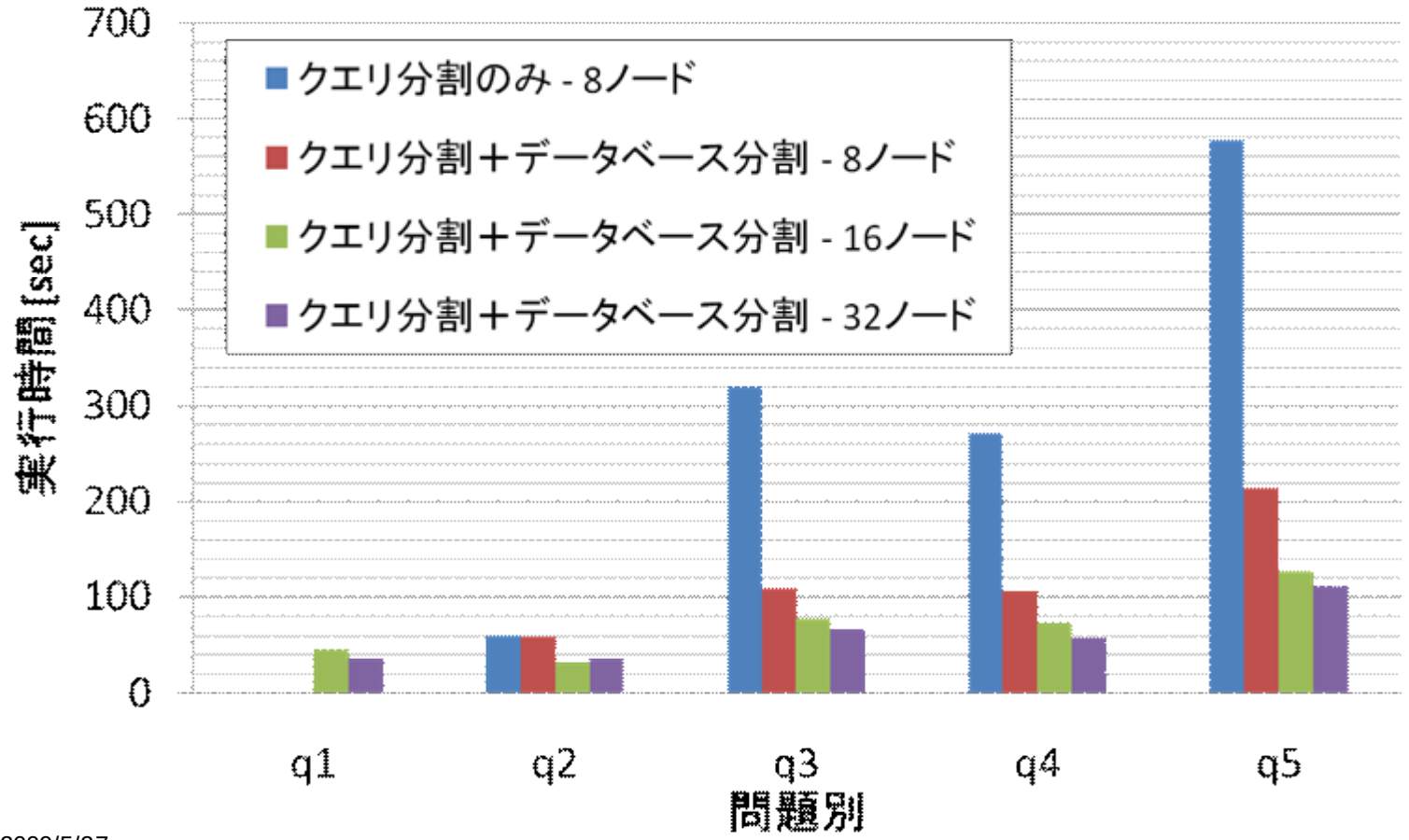
2. コスト (クエリの文字数×データベースの文字数) の大きいクエリのデータベースを分割
   - データベースの分割もリストアルゴリズムによりすべて等負荷になるように
3. 2をコスト最小になるまで繰り返す



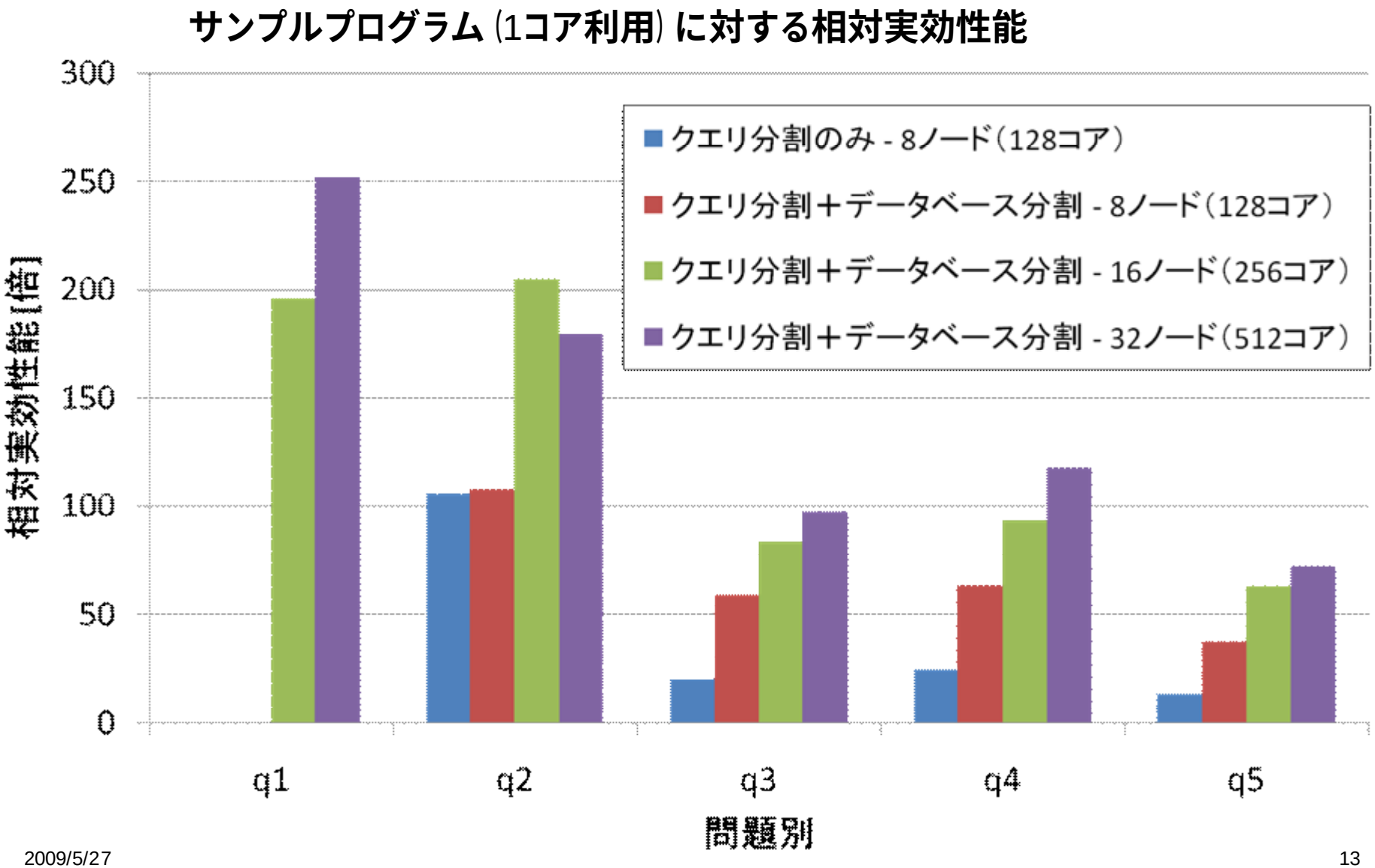
結果 (1)

サンプルプログラムの実行時間 (1コア)

q1	q2	q3	q4	q5
8720.58	6380.30	6496.72	6776.32	8062.42



結果 (2)


サンプルプログラム (1コア利用) に対する相対実効性能
300
250
200
150
100
50
0
相対実効性能 [倍]
クエリ分割のみ - 8ノード (128コア)
クエリ分割＋データベース分割 - 8ノード (128コア)
クエリ分割＋データベース分割 - 16ノード (256コア)
クエリ分割＋データベース分割 - 32ノード (512コア)
q1
q2
q3
q4
q5
問題別

各問題セットの傾向と結果考察

セット		q1	q2	q3	q4	q5
query	*量*	極多	多	並	並	少
	長さ	長	並	並	並	短
database	*量*	並	多	極多	並	少
	長さ	短	並	並	長	極長

- クエリの量が多い問題に弱い
- databaseの量や長さはあまり問題ではない

まとめ･今後の課題

- コアに割り当てるタスクの最小単位はクエリ×データベース
  - クエリ(データベース)の配列数によっては並列性が足りない
  - クエリ(データベース)の文字数によっては十分な高速化が望めない
  - アルゴリズム自体を並列化し，クエリ(データベース)を分割する必要がある
- T2Kアーキテクチャに適した並列化手法･プロセス配置
- コア間通信コストを考慮していないタスクスケジューリング