O-011

# qn24b: N-queens の解を計算するベンチマークプログラム
# qn24b: An N-queens Benchmark Program

吉瀬 謙二 †‡ Kenji Kise　片桐 孝洋 †‡ Takahiro Katagiri　本多 弘樹 † Hiroki Honda　弓場 敏嗣 † Toshitsugu Yuba

## 1. まえがき

N-queens は，互いに攻撃をおこなわないような N 個のクィーンを N × N のボードに配置する解の総数を求める問題である．ここでいうクィーンとはチェスの駒の 1 つの種類のことで，縦，横，斜めの方向に移動することができる．N=6 の問題における 4 つの解を図 1 に示す．

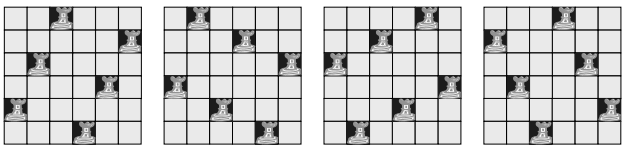

図 1: 6-queens には 4 つの解が存在する．

N-queens は，古くは，チェス盤を利用した 8-queens として 1848 年にチェスプレーヤーの Max Bazzel によって導入された問題とされている [4]．2 年後の 1850 年には Carl Gauss による解法が議論され，その後，多くの数学者の議論の対象となっている．チェス盤のサイズを N とした N-queens 問題に拡張され，近年では，計算機科学の例題として広く取りあげられている．特に，バックトラックを用いたアルゴリズム [5]，分割統治法 [3]，制約充足問題などの例題として利用されている．また，計算機システムなどのベンチマークプログラム [6, 8] としても広く用いられている．

我々は，並列計算機や PC クラスタを含む様々なプラットフォームで利用できるベンチマークとして，N-queens の解を求めるプログラム qn24b を構築した．本稿では qn24b の設計と実装について述べる．また，幾つかの計算機で動作させた結果を報告する．

## 2. qn24b の設計と実装

### 2.1 qn24b の機能

N-queens の解の数を求めるベンチマークプログラム qn24b は，引数として N-queens の問題サイズ N を指定して，計算した解を出力する．また，N=23 までの正しい解の情報を保持しており，これらと，計算により得られた値とを比較することで，計算結果の正しさを確認する．

様々なプラットフォームに対応するために，同一の計算カーネルを用いた逐次版，MPI(Message Passing Interface) を用いた並列版，OpenMP を用いた並列版のコードを提供する．

### 2.2 qn24b の特徴

qn24b は，ベンチマークプログラムに適した多くの特徴を持つ．これらを列挙する．

(1) 計算機システムや並列計算機システムの性能を評価するベンチマークとしては，十分に最適化されたプログラムを利用することが好ましい．qn24b の計算カーネルは，世界で初めて 24-queens の解の数を求めるために最適化を施し PC クラスタを用いて 24-queens の世界記録を計算 [7] したプログラムと同じ最適化されたカーネルを利用しており，ベンチマークプログラムとして適している．(2) 幾つかのベンチマークプログラムには，指定できる台数が 2 のべき乗に制限されるといった問題がある．一方，qn24b では任意の台数を指定することが可能であり，より柔軟なシステム構成の性能を測定できる．(3) メモリシステムの性能を除外した，プロセッサや並列計算機システムの性能評価の一つの指針を提供する．(4) 最もコード量の多い MPI 版でも 300 行程度とコンパクトに記述されており，コード全体を容易に把握できる．(5) 全てのプログラムのコードは 1 つのファイルにまとめられており，容易にコンパイルできる．(6) 問題サイズ N のみがプログラムの入力として利用され，容易に実行と検証をおこなうことができる．(7) 入力パラメータの問題サイズを変化させることで，計算時間を広い範囲で変えることができる．

### 2.3 計算カーネルの実装

高速に動作することを目指して様々な最適化が施されている Jeff Somers[1] のプログラムを参考にして，計算カーネルを作成した．

```
 1 typedef struct array_t{
 2   unsigned int cdt; /* candidates        */
 3   unsigned int col; /* column            */
 4   unsigned int pos; /* positive diagonal */
 5   unsigned int neg; /* negative diagonal */
 6 } array;
 7
 8 long long qn(int n, int h, int r, array *a){
 9   long long answers = 0;
10
11   for(;;){
12     if(r){
13       int lsb = (-r) & r;
14       a[h+1].cdt = (       r & ~lsb);
15       a[h+1].col = (a[h].col & ~lsb);
16       a[h+1].pos = (a[h].pos |  lsb) << 1;
17       a[h+1].neg = (a[h].neg |  lsb) >> 1;
18
19       r=a[h+1].col & ~(a[h+1].pos|a[h+1].neg);
20       h++;
21     }
22     else{
23       r = a[h].cdt;
24       h--;
25
26       if(h==0) break;
27       if(h==n) answers++;
28     }
29   }
30   return answers;
31 }
```

図 2: qn24b の計算カーネル．逐次版と並列版で共通．

†電気通信大学 大学院情報システム学研究科, Graduate School of Information Systems, The University of Electro-Communications
‡独立行政法人科学技術振興機構 さきがけ, PRESTO, Japan Science and Technology Agency(JST)

qn24b のカーネルを図 2 に示す．このコードは逐次版と並列版で利用される．11 行目から 29 行目がメインループで，その中に，12 行目から 28 行目までの if-then-else の簡潔な制御構造を持つ．我々が採用するアルゴリズムでは，N×N のボードにおいて，上段から下方向に順番にクィーンを置いていく (13 行目から 20 行目)．12 行目でクィーンを配置する候補の有無を判定し，候補がない場合にバックトラック (22 行目から 28 行目) する．12 行目でおこなっている候補の判定を予測することは難しい．このため，分岐予測のヒット率はそれほど高くない．

メモリ参照に関しては，11 行目からのループの 1 イタレーションで，array 型の配列 a[h+1] の要素のみが更新されるように工夫している．計算カーネルで参照するメモリ容量は 1KB 以下と非常に小さく，一般的なプロセッサの場合には，非常に高い確率でデータキャッシュにヒットする．このため，メモリシステムのベンチマークとしては適していない．

### 2.4 逐次版の設計と実装

逐次版プログラムのメイン関数を図 3 に示す．N-queens の対称性を利用して，ボードの左半分の解を計算して 2 倍する (19 行目) という高速化を用いている．16 行目で，図 2 に示した計算カーネルを呼び出している．

```
 1 int main(int argc, char *argv[]){
 2   int i;
 3   int n    = set_problem_size(argc, argv);
 4   array *a = calloc(MAX, sizeof(array));
 5   long long usec    = get_time();
 6   long long answers = 0;
 7
 8   for(i=0; i<(n/2+n%2); i++){
 9     long long ret;
10     int h = 1;       /* height or level  */
11     int r = 1 << i; /* candidate vector */
12     a[h].col = (1<<n)-1;
13     a[h].pos = 0;
14     a[h].neg = 0;
15
16     ret = qn(n, h, r, a); /* kernel loop */
17
18     answers += ret;
19     if(i!=n/2) answers += ret;
20   }
21
22   print_result(n, get_time()-usec, answers);
23   return 0;
24 }
```

図 3: 逐次版のメイン関数.

作成した逐次版プログラムと Jeff のプログラムとの動作速度の比較と N-queens の解を表 1 にまとめる．Intel C++ Version 7.1 を用いてコンパイルし，Pentium4 Xeon 2.8GHz の計算機で動作速度 (単位は秒) を測定する．この結果，メモリ参照の効率化と制御構造の最適化を施すことで，qn24b は最大 14%の高速化を達成することを確認した．

### 2.5 OpenMP を用いた並列版の設計と実装

逐次プログラムを並列化するためには，全体の処理を複数のタスクに分割する必要がある．我々が採用するアルゴリズムでは，N×N のボードにおいて，上段から下方向に順番にクィーンを置いていく．この時，最初の数個のクィーンの配置を確定し，残ったクィーンの組み合わせを試しながら解の数をカウントする処理を 1 つのタスクとする．

表 1: N-queens 問題の解と動作速度の比較.

| size N | # of solutions | qn24b (sec) | Jeff (sec) |
|---|---|---|---|
| 15 | 2,279,184 | 1.20 | 1.25 |
| 16 | 14,772,512 | 7.45 | 8.45 |
| 17 | 95,815,104 | 55.56 | 59.52 |
| 18 | 666,090,624 | 384.81 | 440.51 |
| 19 | 4,968,057,848 | 3168.67 | 3417.55 |
| 20 | 39,029,188,884 | 24329.70 | 27745.86 |

このように分割されたそれぞれのタスクは，独立の問題として並列に計算することができる．例えば，6-queens において，最初の 2 個のクィーンの配置が確定したものをタスクとして分割していく場合の 4 つのタスクの様子を図 4 に示す.

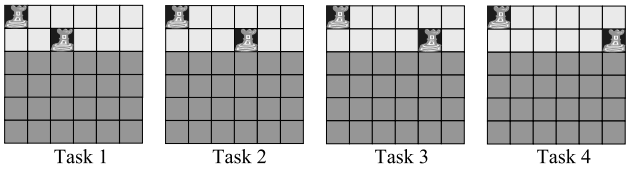


図 4: 6-queens において，最初の 2 個のクィーンの配置を確定してタスク分割をおこなっていく様子.

タスク数と個々のタスクの処理時間を考慮して，最初の 4 個のクィーンの配置が確定した時点からの計算を個々のタスクとする．この時の，問題サイズ N，分割されたタスク数，Pentium4 Xeon 2.8GHz の計算機を用いた場合の 1 つのタスクの平均処理時間 (秒) の関係を表 2 にまとめる．例えば，N=20 の場合に，全ての計算は，平均処理時間 0.78 秒の 30,880 個のタスクに分割される.

表 2: タスク数とタスク当たりの平均計算時間 (秒).

| size N | number of tasks | time (sec)/task |
|---|---|---|
| 16 | 9,844 | 0.0075 |
| 17 | 14,272 | 0.0389 |
| 18 | 18,132 | 0.0212 |
| 19 | 25,080 | 0.1263 |
| 20 | 30,880 | 0.7878 |
| 21 | 41,176 | 6,130 |
| 22 | 49,480 | 45,930 |
| 23 | 64,072 | 354,660 |
| 24 | 75,516 | 3,009,150 |

OpenMP を用いて並列化したメイン関数を図 5 に示す．7 行目の get_sub_problem_num は，分割されたタスクの数を求める関数である．個々のタスクは独立に計算できるので，これらのタスクを OpenMP の parallel for 構文 (10 行目) を用いて並列化する．12 行目の solver と

いう関数が指定した一つのタスクの計算を担当する．

```
 1 int main(int argc, char *argv[]){
 2   int prob[MAX_TASK]; /* store problem IDs */
 3   int rets[MAX_TASK]; /* store each answer */
 4   int i;
 5   int n    = set_problem_size(argc, argv);
 6   int jobs = get_sub_problem_num(n, prob);
 7   long long usec    = get_time();
 8   long long answers = 0;
 9
10 #pragma omp parallel for schedule(dynamic)
11   for(i=1; i<=jobs; i++){
12     rets[i] = solver(n, i, jobs, prob);
13   }
14
15   for(i=1; i<=jobs; i++) answers += rets[i];
16   print_result(n, get_time()-usec, answers);
17   return 0;
18 }
```

図 5: OpenMP を用いた並列化版のメイン関数.

#### 2.5.1 MPI を用いた並列版の設計と実装

MPI を用いた並列化には，割り当てられた仕事が終了した時点で，マスタから仕事を割り当ててもらうマスタ・ワーカ (master-worker) 方式を用いることにする．1 つのプロセスがマスタを担当し，残りのプロセスがワーカとして動作する．

23-queens を例にとると，この問題は 64,072 個のタスクに分割される．測定の結果，それぞれのタスクの処理時間はタスク間で最大 15 倍程度のばらつきがあることが判明した．このように，タスクの計算量が均一ではない場合には，適切なタスク分割とマスタ・ワーカ方式を用いることで高い並列化の効率を期待することができる．

```
 1 typedef struct {
 2   int jno;        /* job number            */
 3   long long ret;  /* the number of answers */
 4   long long usec; /* micro second          */
 5 } packet;
 6
 7 void job_worker(int rank, int n,
 8                 int jobs, int *p){
 9   MPI_Status status;
10   packet send, recv;
11   memset(&send, 0, sizeof(packet));
12   memset(&recv, 0, sizeof(packet));
13
14   while(1){
15     MPI_Sendrecv(&send, sizeof(packet),
16                  MPI_CHAR, 0, TAG_REQ,
17                  &recv, sizeof(packet),
18                  MPI_CHAR, 0, TAG_ACK,
19                  MPI_COMM_WORLD, &status);
20
21     if(recv.jno==0) return;
22
23     send.jno  = recv.jno;
24     send.usec = get_time();
25     send.ret  = solver(n, recv.jno, jobs, p);
26     send.usec = get_time() - send.usec;
27   }
28 }
```

図 6: MPI を用いて記述したワーカの動作.

MPI を用いて記述したワーカのコードを図 6 に示す．マスタから，処理すべきタスク番号を受信し (15 行目から 19 行目)，25 行目の関数 solver が指定した一つのタスクの計算を担当する．また，15 行目から 19 行目の送信において，得られた解の数をマスタに送信する．

## 3. ベンチマーク結果

### 3.1 逐次版のベンチマーク結果

幾つかの計算機で逐次版の動作速度を測定した結果を報告する．利用する計算機環境を列挙する．

**UltraSPARCIIIi** Sun Fire V440, UltraSPARC IIIi 1.062GHz を 4 個搭載，メインメモリ 8GB, SOLARIS 9, gcc -O2 でコンパイル．

**Alpha21264** HP AlphaStation DS10, Alpha21264 600MHz を 1 個搭載，メインメモリ 256MB, Tru64 UNIX 5.1A, Compaq C V6.4 -O2 でコンパイル．

**PentiumIII Xeon** DELL PowerEdge6400, PentiumIII Xeon 700MHz を 4 個搭載，メインメモリ 512MB，RedHat 7.3, icc -O2 でコンパイル．

**Opteron** LIBRAGE TWIN 244, Opteron 244 1.8GHz を 2 個搭載，メインメモリ 2GB, TurboLinux8 for AMD64, gcc -O2 でコンパイル．

**Pentium4 Xeon** DELL PowerEdge2650, Pentium4 Xeon (2.8GHz, 512KB L2 cache) を 2 個搭載，メインメモリ 2GB，RedHat 7.3, icc -O2 でコンパイル．

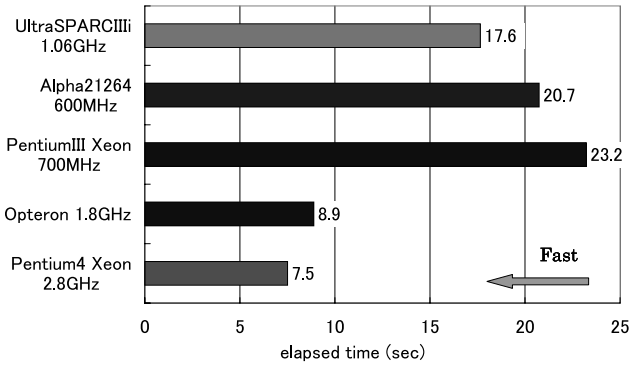


図 7: 逐次版の 17-queens のベンチマーク結果.

17-queens の処理時間の測定結果を図 7 にまとめる．これらの計算機の中では，Pentium4 が最も高い性能を示した．qn24b は，Alpha などの RISC 系の計算機を含む様々なアーキテクチャの計算機で動作する．

### 3.2 OpenMP を用いた並列版のベンチマーク結果

OpenMP を用いた並列版のベンチマーク結果を報告する．16-queens の処理時間と，1 スレッドの場合を基準とした速度向上率を表 3 にまとめる．Intel C++ Version 7.1 を用いてコンパイルし，4 個の PentiumIII Xeon 700MHz のプロセッサを搭載する SMP 型の計算機 DELL PowerEdge6400 を利用した．

表 3 にまとめた結果から，スレッド数を増やすに従って，スレッド数と同様の理想的な速度向上が得られることを確認できる．

表 3: OpenMP を用いた並列版の速度向上.

| number of threads | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| time (sec) | 21.83 | 10.92 | 7.30 | 5.47 |
| speedup | 1.00 | 2.00 | 2.99 | 3.99 |

### 3.3 MPI を用いた並列版のベンチマーク結果

MPI を用いた並列版のベンチマーク結果を報告する．Intel C++ Version 7.1 を用いてコンパイルし，34 ノード，68CPU から構成される PC クラスタ [9] で 20-queens の処理速度を測定した．ノード間はギガビットのイーサネットで接続されている．個々のノードは 2 個の Pentium4 Xeon 2.8GHz のプロセッサを搭載する SMP 型の計算機で，Hyper-Threading を有効にしている．このため，ノード当たり 4 つの論理プロセッサが存在し，それに対応する 4 つのプロセスを生成している．図 8 は，1 ノードの処理時間を基準とした速度向上率である．

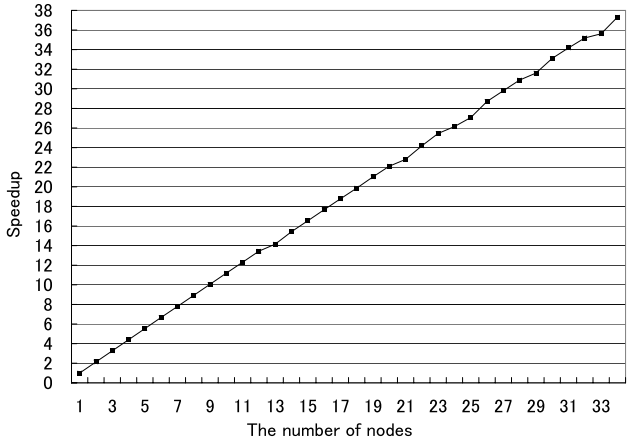


図 8: MPI を用いた並列版の速度向上.

図 8 にまとめた測定結果から，ノード数の増加に比例する性能向上が得られることを確認できる．

これらの測定結果が示すように，並列版の qn24b は，プロセッサ数やノード数に比例して性能が向上する．このため，SMP や PC クラスタの動作テストとしての利用にも適している．

## 4. おわりに

並列計算機や PC クラスタを含む様々なプラットフォームで利用できるベンチマークとして，N-queens の解の数を求めるプログラム qn24b を構築した．本稿では，qn24b の設計と実装を議論するとともに，幾つかの計算機で動作させた結果を報告した．

qn24b は，引数として N-queens の問題サイズ N を指定して，計算した解の数を出力する．また，N=23 までの正しい解の情報を保持しており，これらと計算により得られた値とを比較することで，計算結果を検証できる．様々なプラットフォームに対応するために，同一の計算カーネルを用いた逐次版，MPI を用いた並列版，OpenMP を用いた並列版のコードを提供する．

qn24b は，ベンチマークプログラムに適した多くの特徴を持つ．(1) 計算カーネルとして，最適化された高速なものを利用している．(2) 様々なシステム構成の性能を測定できるように，任意の台数で動作させることができる．(3) メモリシステムの性能を除外した，主にプロセッサや並列計算機システムの性能評価の一つの指針を提供する．(4) コンパクトに記述されており，コード全体を容易に把握できる．(5) 全てのプログラムのコードは 1 つのファイルにまとめられており，容易にコンパイルできる．(6) 問題サイズ N のみがプログラムの入力として利用され，容易に実行と検証をおこなうことができる．(7) 入力パラメータの問題サイズを変化させることで，計算時間を広い範囲で変えることができる．

qn24b のソースコードは次の URL からダウンロードできる．

http://www.yuba.is.uec.ac.jp/%7kis/nq/

## 参考文献

[1] The n queens problem, a study in optimization. http://www.jsomers.com/.

[2] The on-line encyclopedia of integer sequences. http://www.research.att.com/ njas/sequences/.

[3] Bruce Abramson and Moti Yung. Divide and conquer under global constraints: a solution to the n-queens problem. *J. Parallel Distrib. Comput.*, 6(3):649–662, 1989.

[4] Cengiz Erbas, Seyed Sarkeshik, and Murat M. Tanik. Different perspectives of the n-queens problem. In *ACM annual conference on Communications*, pages 99–108, 1992.

[5] Solomon W. Golomb and Leonard D. Baumert. Backtrack programming. *J. ACM*, 12(4):516–524, 1965.

[6] J. R. Gurd and D. F. Snelling. Manchester dataflow: a progress report. In *Proceedings of the 6th international conference on Supercomputing*, pages 216–225, 1992.

[7] Kenji Kise, Takahiro Katagiri, Hiroki Honda, and Toshitsugu Yuba. Solving the 24-queens problem using mpi on a pc cluster. Technical Report UEC-IS-2004-6, Graduate School of Information Systems, The University of Electro-Communications, June 2004.

[8] Michael D. Noakes, Deborah A. Wallach, and William J. Dally. The j-machine multicomputer: an architectural evaluation. In *Proceedings of the 20th annual international symposium on Computer architectur e*, pages 224–235, 1993.

[9] 吉瀬 謙二, 片桐 孝洋, 本多 弘樹, and 弓場 敏嗣. Clustermatic を用いた PC クラスタの構築. Technical Report UEC-IS-2004-3, 電気通信大学 大学院情報システム学研究科, March 2004.