PEMROSESAN PARALEL ALGORITMA A* DAN OPTIMAL MENGGUNAKAN *MESSAGE PASSING* UNTUK PELACAKAN DALAM RUANG MASALAH *TREE*: STUDI KASUS *PUZZLE* KOTAK–8

Agust Isa Martinus
Teknik Informatika, Fakultas Teknik, Universitas Muhammadiyah Cirebon
<agust.isa@umc.ac.id>

Abstrak

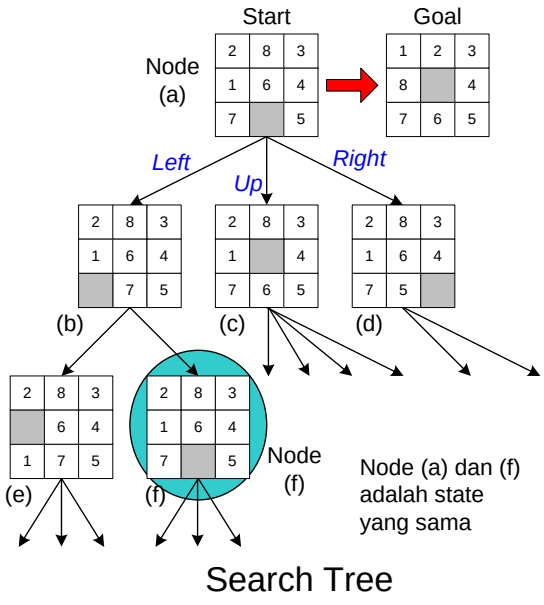
Kecepatan clock komputer masa kini sudah mencapai batas yang diprediksi oleh Amdahl (Amdahl's Law), yaitu pada kisaran 3 GHz sehingga sebelum ditemukan dan diterapkan teknologi baru untuk pembuatan komputer, maka clock komputer akan sulit untuk ditingkatkan lagi secara signifikan. Untuk memenuhi kebutuhan peningkatan produktifitas sistem komputer, salah satu hal yang dapat ditingkatkan adalah throughput sistem komputer, bukan hanya kecepatan clock-nya. Peningkatan throughput suatu sistem komputer dapat dicapai dengan menerapkan pipeline, hyperthread, multicore, parallel computer, atau distributed computer system yang lainnya. Selain perangkat keras yang memadai, tentunya diperlukan juga sistem operasi, program-program, dan algoritma-algoritma yang dapat mendukung perangkat keras yang ada dalam peningkatan kinerja sistem komputer. Untuk mendapatkan pemahaman lebih mengenai pemrosesan paralel, dalam penelitian ini dilakukan percobaan paralelisasi dan/atau distribusi proses secara eksplisit. Program dibuat dalam Bahasa C dengan library message passing MPICH kemudian diujikan metoda paralelisasi dan distribusi Message Passing untuk algoritma pelacakan A (A-star) dan Optimal (Dijkstraa) dengan representasi masalah dalam Tree untuk menyelesaikan masalah Puzzle Kotak-8. Hasil eksekusi beberapa pengujian yang dilakukan, dirangkumkan dalam beberapa tabel. Pengamatan dilakukan pada masalah kecepatan (speed up dan relative speed up yang merupakan parameter time complexity) pada jumlah proses mulai dari 1-proses, 2-proses, sampai dengan 128-proses. Dari hasil pengujian didapatkan beberapa hal, ada percepatan proses dan ada juga perlambatan proses, misalkan pelacakan Optimal dengan kedalaman 22-level terjadi percepatan untuk semua jumlah proses mulai 2, 4, 8, s.d. 128-proses, dan percepatan tertinggi pada 16-proses dengan percepatan sekitar 360 kali dibandingkan kecepatan (throughput) 1-proses. Pada pelacakan Optimal 18-level terjadi percepatan sekitar 7 s.d. 8 kali untuk jumlah proses 2, 4, 8, 16-proses, 3 kali pada 32-proses, 2 kali pada 64-proses, tetapi untuk jumlah 128-proses terjadi perlambatan (0,5 x throughput 1-proses). Sedangkan pada pelacakan Optimal 9-level, terjadi perlambatan untuk semua jumlah proses yang diujikan. Demikian halnya dengan pelacakan menggunakan algoritma A*, terjadi perlambatan untuk semua jumlah proses yang diujikan.*

***Kata kunci**: Optimal Search, A* Algorithm, Message Passing, Sistem Terdistribusi, Pemrograman Paralel*

PENDAHULUAN

Berbagai cara paralelisasi dan distribusi pada komputasi komputer sudah ditemukan dan dikembangkan oleh para ahli. Berikut digunakan metoda paralelisasi dan distribusi *Message Passing* (menggunakan program MPICH) untuk dua algoritma program pelacakan A* (*A-star*) dan Optimal (Dijkstraa) dalam *Tree* (pohon masalah) pada permasalahan yang sudah dikenal, yaitu masalah *Puzzle* Kotak-8.

Kemampuan komputasi dari komputer masa kini sudah sangat tinggi (*power full*). Hal ini dikarenakan kecepatan *clock* semakin tinggi, L1 dan L2 *cache memory* yang semakin cepat dan semakin besar kapasitasnya, *hyperthread* untuk tiap *core*, jumlah *core* (*microprocessor*) yang lebih dari satu dalam satu keping pemroses, sampai dengan penyediaan *bus-bus* khusus terpisah untuk masing-masing keperluan lalu-lintas data atau informasi di dalam sistem komputer. Kemampuan komputasi perangkat keras komputer tersebut didukung oleh kemampuan sistem operasi terkini yang memungkinkan *multithreading* sampai dengan *multiprocessing* dan/atau pemrosesan paralel sehingga semakin menambah daya komputasi sistem komputer. Selain itu, perangkat dan arsitektur serta perangkat lunak pendukung



Gambar 1. Pelacakan Pada Puzzle Kotak-8

jaringan komputer terkini juga sudah sangat memungkinkan direalisasikan dan digunakan secara nyaman untuk komputasi paralel, HPC (*High Performance Computer*), atau komputasi *Grid*, sampai dengan sistem komputer terdistribusi.

Sebagian pengguna komputer atau gawai (*gadget*) tidak menyadari bahwa proses yang dijalankannya dalam perangkatnya, dikerjakan secara paralel oleh perangkat komputasi tersebut (paralel secara implisit). Sebagian lagi yang membutuhkan kinerja komputasi tinggi (para saintis, mesin enkripsi dan dekripsi, rendering gambar atau video, mesin peramalan cuaca, dsb.) biasanya melakukan paralelisasi secara eksplisit sehingga dapat sesuai dengan yang dirancang. Dalam kesempatan ini, akan dilakukan paralelisasi atau distribusi proses secara eksplisit menggunakan *message passing* untuk menyelesaikan permasalahan pelacakan pada *Puzzle* Kotak-8 dengan algoritma pelacakan A* dan Optimal.

Algoritma A*, Optimal, dan *Best First Search* menggunakan alur algoritma yang sama. Perbedaan ketiganya hanya terletak pada parameter untuk fungsi evaluasinya. Pada Algoritma A* melibatkan nilai *heuristic* dan "*real cost*" untuk nilai parameter pada fungsi evaluasinya. Pada Algoritma Optimal hanya melibatkan "*real cost*"-nya. Sedangkan pada *Best First Search* hanya melibatkan "*heuristic*"-nya. Karena pada masalah *Puzzle* Kotak-8, nilai untuk semua cabangnya dari *node* n_i ke n_{i+1} adalah sama yaitu 1 (satu langkah), maka nilai "*real cost*"-nya adalah level atau kedalaman dari cabang pohon (*node* n_i) yang dimaksud dari *initial state*. Sedangkan nilai *heuristic*-nya adalah total jumlah dari kotak-kotak yang tidak pada tempatnya (*unmatch*) ditambah jumlah jarak (*manhattan distance*) dari semua *unmatch* ke tempat semestinya. Ketiga algoritma tersebut ditunjukkan pada Progam 1.

Penilaian kinerja suatu algoritma pelacakan, setidaknya dilihat dari empat parameter, yaitu *completeness*, *optimality*, *time complexity*, dan *space complexity*. Algoritma pelacakan Optimal dan A* merupakan algoritma yang *complete* (dapat menemukan *goal state*) dan *optimal* (*goal state* yang ditemukan berbiaya optimal). Pada kesempatan ini, sesuai dengan tujuan umum paralelisasi suatu sistem, kita akan mengukur kecepatan (*time complexity*), terutama perbandingan kecepatan paralelisasi dengan kecepatan proses tunggal.

Pelacakan untuk memperoleh Solusi Optimal

- Mencari jejak dengan 'cost' (biaya) minimum
- 'Cost' untuk penerapan fungsi evaluasi diketahui

Prosedur Pelacakan

1. **- Berikan node awal S pada OPEN,** //initial state
   - eVal(s) = 0 //fungsi evaluasi
2. **Loop: if OPEN==kosong then exit (fail)** //solusi tidak ditemukan
3. **n := first (OPEN)** //head dari Priority Queue
4. **if GOAL(n) then exit (success)**
5. **Remove (n, OPEN)** //hapus dari Priority Queue
6. **- Ekspansikan n,**
   - hitung eVal(n_i) untuk semua node anak n_i
   - dan bubuhkan pointer dari n_i ke n.
   - Berikan semua simpul anak pada OPEN dan urutkan mulai eVal(n_i) terendahnya
   //membentuk Priority Queue
7. **Kembali ke Loop** //kembali ke langkah 2

Penjelasan

- *n* adalah node yang sedang dievaluasi
- OPEN adalah struktur data (ADT) berbentuk Priority Queue.
- *eVal*(n_i) adalah fungsi evaluasi yang menentukan biaya:
  - Optimal: $eVal(n_i) = g'(n_i)$
  - A*: $eVal(n_i) = g'(n_i) + \hat{h}(n_i)$
- $g'(n_i)$ adalah fungsi (total) biaya dari initial state sampai dengan node n_i.
- $\hat{h}(n_i)$ adalah fungsi heuristic (unmatch + manhattan) dari node n_i.

Progam 1. Algoritma Pelacakan A* dan Optimal

PROSES PARALELISASI DAN DISTRIBUSI

Paralelisasi dan distribusi yang dirancang (untuk kemudahan) menggunakan proses sejumlah perpangkatan 2 (2^n, dengan n=0, 1, 2, 3,

...). Program akan mengalami "*error*" atau "*hang*" jika jumlah proses yang terlibat bukan perpangkatan 2 karena algoritma penentuan alamat (nomor proses/*rank*) yang dibuat, akan menghasilkan nomor proses (*source* atau *destination*) yang tidak ada dalam MPI_COMM_WORLD jika jumlah proses bukan perpangkatan 2. Program akan *error* jika mencoba mengirim ke/nenerima dari proses yang tidak ada. Sedangkan program akan mengalami "*hang*" karena ada beberapa proses yang menunggu *message* dari proses lain, tetapi tidak ada yang mengirimkan *message* padanya.

Prosedur Pelacakan Secara Paralel

1. **if proses 0**
   - Langsung ke langkah 2
   else
   - Tunggu initial node //MPI_Recv(...)
2. - Berikan initial node S pada OPEN, //initial state
   - eVal(s) = 0 //fungsi evaluasi
3. Loop: if OPEN==kosong then exit (fail) //solusi tidak ditemukan
4. n := first (OPEN) //head dari Priority Queue
5. if GOAL(n) then exit (success)
6. Remove (n, OPEN) //hapus dari Priority Queue
7. - Ekspansikan n,
   - hitung eVal(n_i) untuk semua node anak n_i
   - dan bubuhkan pointer dari n_i ke n.
   - Berikan semua simpul anak pada OPEN dan urutkan mulai eVal(n_i) terendahnya //membentuk Priority Queue
8. **while OPEN > 1 && tujuan belum semua terkirimi**
   - **n := first (OPEN) //head dari Priority Queue**
   - **Kirim node n ke proses tujuan berikutnya //MPI_Isend(...)**
   - **Remove (n, OPEN) //hapus dari Priority Queue**
9. Kembali ke Loop //kembali ke langkah 3

Progam 2. Paralelisasi Algoritma Pelacakan A* dan Optimal

Untuk pemrosesan program secara paralel, dilakukan modifikasi algoritma Progam 1. dengan menambahkan/menyisipkan beberapa instruksi seperti ditunjukkan pada Progam 2 (instruksi no. 1 dan 8) yang diimplementasi pada penggalan Program 3 dan Program 4.

IMPLEMENTASI PROGRAM DAN STRUKTUR DATA YANG DIGUNAKAN

Implementasi program pelacakan dengan Algoritma A* dan Optimal menggunakan Bahasa Microsoft Visual C++ versi 6.0 dan *library* untuk *message passing* dari MPICH versi 1.25. Sedangkan struktur data yang diimplementasi ditunjukkan pada *snippet* (dalam *header file* K8Tree.h). Untuk implementasi program selengkapnya dapat dilihat pada lampiran Source Code Program yang ada pada penulis.

```
//...
while ((openCount>1) && (fold>=1) &&
hOpen) {
        j = myrank;
        pes = myrank + fold;
        while (j < pes) {
                j += fold;

                if ((j - fold) == myrank)
dest = j;
                j += fold;
        }
        fold /=2;

        //... di sini inisialasasi
parameter yang akan dikirim

//kirim initial state ke proses tetangga
//      MPI_Send(val, 6, MPI_LONG, dest,
tag++, MPI_COMM_WORLD);
        MPI_Isend(val, 6, MPI_LONG, dest,
tag++, MPI_COMM_WORLD, &request);

        if (hOpen) {
                old = hOpen;
                hOpen = hOpen->next;
                delete old;
        //hapus dari Open List
                openCount--;
        }
}
//...
```

Program 3. Proses Pengiriman Message dan Penentuan Alamat Penerima

Pengiriman *message* menggunakan metoda "*non-blocking*" karena proses-proses pengirim akan mengirimkan *message* ke satu penerima tertentu hanya sekali, jadi tidak akan terjadi penimpaan *message* sebelumnya. Implementasi penentuan alamat pengirim dan proses menunggu penerimaan *message* (*initial node*) dari proses "tetangganya" ditunjukkan pada penggalan

```
//...
while (fold >= 1) {
        j=0;
        while (j < npes) {
                j += fold;

                if (j == myrank) {
                        source = j - fold;
                        myfold = fold/2;
                }
                j += fold;
        }
        fold /=2;
}
//tunggu initial state dari proses tetangga
MPI_Recv(val, 6, MPI_LONG, source,
MPI_ANY_TAG, MPI_COMM_WORLD, &status);
//...
```

Program 4. Proses Penerimaan Message dan Penentuan Alamat Pengirim

Program 4. Proses menunggu *message* menggunakan metoda "*blocking*" karena proses ini tidak akan melakukan pekerjaanya sebelum mendapatkan *initial node* untuk diekspansi, dan proses ini hanya menunggu satu initial node,

Dalam implementasi, kita gunakan angka 0 untuk mewakili ubin kosong yang dapat digerak-gerakkan. Ubin-ubin dalam Kotak-8 diberi koordinat (x,y) dari (0,0) s.d. (2,2) untuk menyatakan posisinya dalam Kotak-8. Sebagai contoh, *Goal State* pada Gambar 1, koordinat (0,0)=1, (0,1)=2, (0,2)=3, (1,0)=8, (1,1)=0, (1,2)=4, (2,0)=7, (2,1)=6, dan (2,2)=5.

Struktur data yang diimplementasi terdiri dari tipe data `TILE` untuk menangani satu kotak yang berisi satu angka unik dari 0 s.d. 8. Data tipe `TILE` menunjukkan posisi suatu ubin (angka) dalam Kotak-8, misalnya `goalTile[5].x` berisi 1 dan `goalTile[5].y` berisi 2 artinya angka 5 berada pada posisi (1,2) dalam Kotak-8. Tipe

```
//...
struct TILE {
        int x,y;
}goalTile[9], nodeTile[9];

struct NODE {
        long int id;
        ARAH arah;      //arah sebelumnya,
jk sblmny UP -> DOWN jgn dlakukn
        NODE* parent; //link ke node di
atasnya dlm Tree/Graph
        NODE* path; //link k node d bwhny
dl Tree/Graph yg tmsk Sol Path
//      NODE* prev; //u/ link node depan
dlm Queue yg sudah dibentuk
//      NODE* next; //u/ link node belakang
dlm Queue yg sudah dibentuk
        int eVal;       //nilai yg
dievaluasi
        int manhattan;
        int unmatch;
        int depth;
        bool opening;
}goalNode, isNode, *head, *tail;

struct ORDLIST {        //struktur untuk
Priority Queue
        ORDLIST *next;
        NODE *node;
}*hOpen, *tOpen;
//...
```

Program 6. Struktur Data Program Pelacakan A* dan Optimal

data berikutnya adalah `NODE` yang berfungsi untuk menangani satu keadaan (*state* atau *node*) dari Kotak-8. Satu keadaan merupakan satu kemungkinan permutasi dari angka-angka pada ubin dalam Kotak-8. Tipe data `NODE` memuat data-data yang dibutuhkan oleh suatu keadaan atau node.

Tipe yang terakhir, `ORDLIST`, adalah ADT (*Abstract Data Type*) yang berfungsi untuk membentuk antrian terurut (*Priority Queue*). Tipe data `ORDLIST` ini untuk menampung anak-anak cabang hasil ekspansi suatu node. Penyimpanan

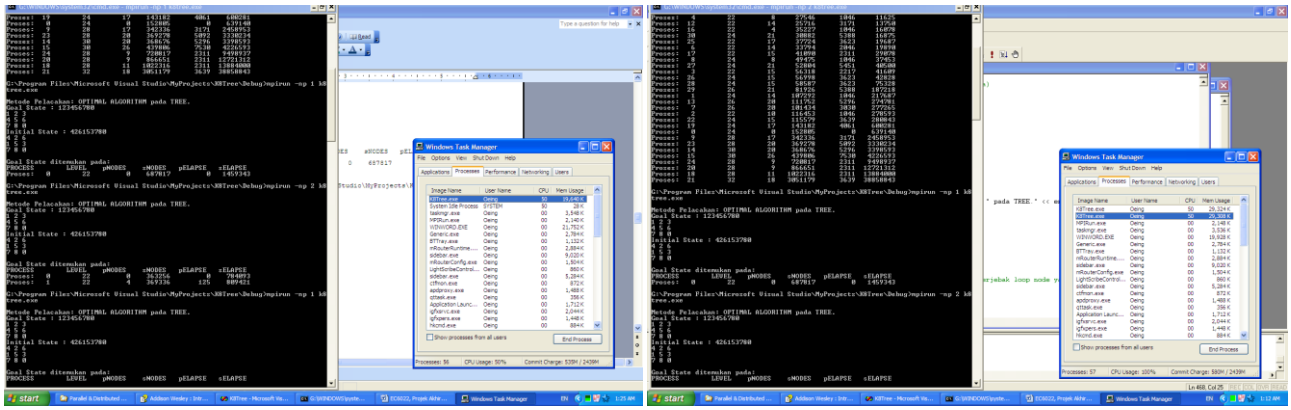
```
.....
//    method=BEST;
      method=OPTIMAL;
//    method=ASTAR;
.....
      gs = 123456780; //Goal State
//    gs = 123804765; //Goal State
//    gs = 123804567; //Goal State
.....
//initial state
//    is = 158203647;    //tanpa solusi
//    is = 123460758;
//    is = 123405786;
//    is = 123485706;
//    is = 235180476; //mulai dari sini,
pada BEST akan terjebak loop node yang
pernah diekspansi
//    is = 283156470;
//    is = 463152078;
//    is = 463572108;
//    is = 460523178;
//    is = 426573108;
//    is = 426713580;
//    is = 412706583;
//    is = 412763580;
//    is = 412063758;
//    is = 460123758;
      is = 426153780;    //OPTIMAL: 22-
levels & 690313-nodes; A*: 22-levels &
3186-nodes
//    is = 26413758;
//    is = 201736548;
//    is = 231576048;
//    is = 263741058;
//    is = 315628470;
//    is = 15628473;     //tanpa solusi
.....
```

Program 5. Initial State dan Goal State yang Digunakan dalam Pengujian

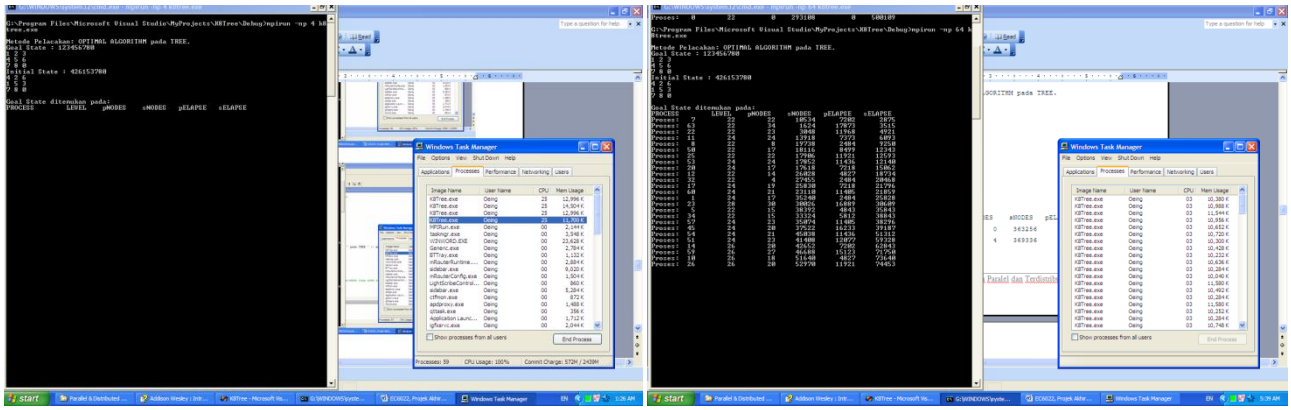
ke dalam `ORDLIST` tersusun secara meningkat (*ascending*). `ORDLIST` berisi informasi urutan node-node yang akan (tapi belum) dievaluasi. Sedangkan node-node yang harus dievaluasinya terdapat dalam tipe data `NODE` yang ditunjuk oleh `var->node`, dengan `var` adalah *pointer* ke tipe data `ORDLIST`.

HASIL EKSEKUSI PROGRAM DAN PEMBAHASAN

Program Pelacakan A* dan Algoritma pada Masalah Puzzle Kotak-8 dijalankan dengan mencoba berbagai jumlah proses (2^n) dari 1, 2, 4, 8, ..., 128. Pada percobaan menjalankan dengan 256 proses, tampil pesan kesalahan yang menyatakan bahwa sistem kekurangan *buffer memory* untuk menjalankan program. Program dijalankan dalam lingkungan mesin dengan spesifikasi sebagai berikut:



(a) 1-Proses **(b) 2-Proses**



(c) 4-Proses **(d) 64-Proses**

Gambar 2. Windows Task Manager dengan K8Tree Berbagai Proses

Processor : Intel Centrino Duo
Core Speed : 2.2 GHz, 2.2 GHz
L2 Cache : 4 Mbytes (2 x 2 Mbytes)
Main Memory : 1 GBytes
Operating System : Windows XP Pro SP2
MPICH : Version 1.25

Pada saat program dijalankan menggunakan 1 proses, *utilitas* prosesor yang digunakan oleh program sekitar 50%. Dengan jumlah 2 proses, utilitas prosesor 100% dengan masing-masing proses mendapatkan sekitar 50%. Dengan jumlah 4 proses, utilitas prosesor 100% dengan masing-masing proses mendapatkan sekitar 25%. Untuk jumlah proses 2 atau lebih, utilitas prosesor 100% dengan masing-masing proses mendapatkan sekitar *100%/jumlahproses* (100% dibagi jumlaproses). Utilitas prosesor sekitar *100%/jumlahproses* untuk masing-masing proses tersebut didapatkan pada saat *peak* (semua proses sedang/masih aktif melakukan pelacakan). Utilitas prosesor untuk masing-masing proses yang lebih tepat adalah *100%/jmlhprosesaktif* (jmlhprosesaktif adalah banyaknya proses yang masih melakukan pelacakan). Untuk melihat fenomena tersebut, dapat dilihat pada gambar-gambar Windows Task Manager yang menunjukkan proses-proses yang aktif dengan

Tabel 1. Ringkasan Proses Tercepat Pelacakan Optimal 22 Level

No	Jml Pros	No. Pros	Level	Parent Nodes	Nodes	Total Nodes	Parent Elapse	Elapse	Total Elapse	Speed Up	S.U. Rel	Spdup/ Pros
1	**1**	0	22	0	687817	**687817**	0	1450343	**1450343**	**1**	-	1
2	**2**	0	22	0	363256	**363256**	0	784093	**784093**	**1.861**	**1.861**	0,9249
3	**4**	1	22	8	76474	**76482**	156	29328	**29484**	**49.496**	**26.594**	12,298
4	**8**	3	22	14	41248	**41262**	702	8500	**9202**	**158.590**	**3.204**	19,701
5	**16**	5	22	17	16992	**17009**	937	3125	**4062**	**359.267**	**2.265**	22,316
6	**32**	11	22	23	3908	**3931**	5374	2265	**7639**	**191.038**	**0.532**	5,933
7	**64**	7	22	22	10534	**10556**	7202	2875	**10077**	**144.819**	**0.758**	2,249
8	**128**	44	22	23	2191	**2214**	18639	6656	**25295**	**57.692**	**0.398**	0,448

utilitas prosesor masing-masing.

Program dijalankan dengan berbagai *jumlah proses*, *initial state dan goal state*, dan *algoritma A* maupun Optimal*. Berikut ini ditabelkan ringkasan dari hasil eksekusi pelacakan Optimal yang menghasilkan kedalaman optimal 22 level dengan mencatat proses tercepatnya untuk masing-masing jumlah proses (Tabel 1). Kolom *Speed Up* adalah hasil perbandingan total waktu dibagi total waktu untuk 1-proses (total elapse/(*total elapse* baris-1)). Sedangkan kolom *S.U. Rel* (*Relative Speed Up*) total waktu dibagi total waktu jumlah proses setengahnya ((*total elapse*) dibagi (total elapse baris sebelumnya)), kecuali jumlah proses satu tidak ada relatif terhadap jumlah proses ½.

Gambar 3. Perbandingan Speed Up dengan Jumlah Proses, Optimal 22 Level

Tabel 2. Ringkasan Proses Tercepat Pelacakan Optimal 9 Level

No	Jml Pros	No. Pros	Level	Parent Nodes	Nodes	Total Nodes	Parent Elapse	Elapse	Total Elaps	Speed Up	S.U. Rel	Spdup/ Proses
1	**1**	0	9	0	966	**966**	0	31	**31**	**1**	-	1
2	**2**	0	9	0	725	**725**	0	140	**140**	0.2214	0.2214	0,11071
3	**4**	1	9	6	239	**245**	437	265	**702**	0.0442	0.1994	0,01104
4	**8**	2	17	6	9226	**9232**	437	718	**1155**	0.0268	0.6078	0,00336
5	**16**	4	17	6	6593	**6599**	484	453	**937**	0.0331	1.2327	0,00207
6	**32**	12	15	12	913	**925**	2218	1218	**3436**	0.0090	0.2727	0,00028
7	**64**	24	15	12	430	**442**	5874	2937	**8811**	0.0035	0.3900	5,5E-05
8	**128**	40	15	12	190	**202**	11968	6109	**18077**	0.0017	0.4874	1,34E-05

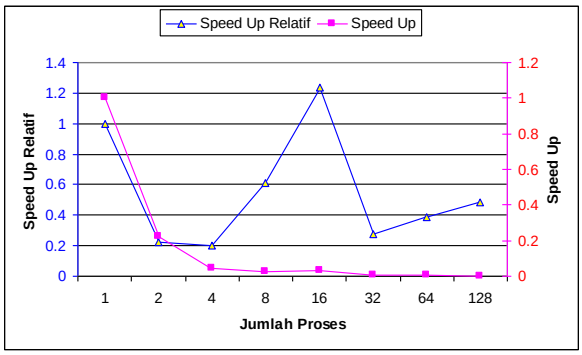
Tabel 1 hanya menampilkan proses tercepat pada masing-masing eksekusi dengan sejumlah proses. Selain *waktu tercepat*, ada juga *jumlah node paling sedikit* yang tidak selalu bersesuaian dengan *waktu total* (*total elapse time*). Hal tersebut karena walaupun jumlah *node* yang dibentuk lebih sedikit, tetapi proses tersebut harus menungu beberapa "*hop*" sebelum diinisiasi (mendapatkan *initial node*) sehingga waktu tunggu (*parent elapse*) lebih lama. Gambar 3 adalah grafik yang dibuat berdasarkan Tabel 1.

Pada grafik Gambar 3 terlihat peningkatan *speed up* (*Relative Speed Up*, dibanding jumlah proses setengahnya) tidak berupa perpangkatan dari 2 (kelipatan 2, untuk *relative speed up*), hal tersebut bukan karena anomali, tetapi karena sifat alamiah dari masalah pelacakan *Puzzle* Kotak-8 yang tidak dapat diketahui (dari awal) jumlah keseluruhan *node* sehingga distribusi banyaknya *node* dari *initial state* sampai dengan ditemukannya *goal state* untuk masing-masing proses tidak merata. (Berbeda dengan perkalian *matrix* yang dapat didistrubsikan secara merata dari sejak awal.) Terlihat juga ada terjadi *relative speed up* < 1 setelah jumlah proses lebih dari 16, hal ini bisa terjadi karena ketersedian (*utility*) prosesor yang didapatkan oleh masing-masing proses semakin kecil, tidak lagi kecepatan penuh untuk masing-masing proses tersebut (dapat dilihat pada Gambar Windows Task Manager).

Walau demikian, paralelisasi pelacakan algoritma Optimal untuk kedalaman optimal 22 level, mulai dari jumlah 2 proses sampai dengan 128 proses, semuanya mengalami speed up > 1. Speed up tertinggi terjadi pada jumlah proses 16, yaitu sekitar 360 kali kecepatan proses tunggal.

Tabel 2 adalah rangkuman dari hasil eksekusi pelacakan Optimal yang menghasilkan kedalaman optimal 9 level pada 1 proses dengan melihat proses tercepatnya untuk masing-masing jumlah proses.

Tabel 3 adalah rangkuman hasil eksekusi algoritma Optimal untuk berbagai jumlah proses



Gambar 4. Perbandingan Speed Up dengan Jumlah Proses, Optimal 9 Level

dengan waktu tercepat (*total elapse time*) hasil

Tabel 3. Ringkasan Proses Tercepat Pelacakan Optimal 18 Level

No	Jml Pros	No. Pros	Level	Parent Nodes	Nodes	Total Nodes	Parent Elapse	Elapse	Total Elaps	Speed Up	S.U. Rel	Spdup/ Proses
1	**1**	0	18	0	94260	**94260**	0	10546	**10546**	**1**	**-**	1
2	**2**	0	18	0	35391	**35391**	0	1203	**1203**	8,7664	8,7664	4,383
3	**4**	0	18	0	27592	**27592**	0	1468	**1468**	7,1839	0,8195	1,796
4	**8**	3	20	14	13918	**13932**	452	937	**1389**	7,5925	1,0569	0,949
5	**16**	1	18	10	11972	**11982**	468	765	**1233**	8,5531	1,1265	0,535
6	**32**	2	18	10	9340	**9350**	921	2421	**3342**	3,1556	0,3689	0,099
7	**64**	4	18	10	6696	**6706**	2703	2671	**5374**	1,9624	0,6219	0,031
8	**128**	8	20	10	11908	**11918**	6578	13078	**19656**	0,5365	0,2734	0,004

eksekusi pelacakan Optimal yang menghasilkan kedalaman optimal 18 level. Dari tabel tersebut kita dapatkan bahwa paralelisasi proses mengakibatkan speed up > 1, kecuali untuk jumlah proses 128 atau lebih.

Untuk pelacakan yang menghasilkan jumlah node relatif sedikit, paralelisasi dengan metoda *message passing* tidaklah cocok. Pada Tabel 2 dan Gambar 4 dapat dilihat bahwa selain percepatan yang lebih kecil dari 1 (Speed Up < 1), juga waktu tercepatnya tidak selalu didapat pada kedalaman optimalnya, atau, kedalaman optimal tidak selalu didapat dengan waktu tercepat. Hal ini dikarenakan waktu yang dibutuhkan untuk mengirim dan menerima *message* jauh lebih lama dibandingkan dengan waktu pelacakannya sendiri.

Hasil pengujian menggunakan algoritma pelacakan A*, kami sajikan dalam Tabel 4 yang merupakan rangkuman proses tercepat untuk kedalaman optimal 22 level jika dilacak menggunakan satu proses. Dari Tabel 4 kita dapat mengetahui bahwa terjadi perlambatan (*Speed up* < 1). Hal tersebut karena pada algoritma pelacakan A* terjadi pemangkasan (*prunning*) *node* yang dihasilkan (diekspansi) akibat adanya informasi *heuristic* sehingga jumlah *node* yang dilacak menjadi sangat sedikit (jika dibandingkan dengan algoritma Optimal) untuk masing-masing proses. Akibat lain dari jumlah *node* yang sedikit adalah waktu tercepat menemukan *goal state* dicapai oleh proses nomor 0 (nol, proses awal) karena waktu pelacakan menuju *goal state* pada masing-masing proses tidak jauh berbeda, tetapi proses nomor 0 tidak perlu menunggu *message* untuk inisiasi (kiriman *initial state*).

PENUTUP

Paralelilasasi proses atau program komputer, jelas dapat dilakukan baik secara implisit maupun eksplisit. Dengan paralelisasi proses, kita bisa mendapatkan peningkatan kecepatan (*throughput*), tetapi tidak selalu terjadi *speed up* di atas 1 (bisa terjadi perlambatan). Peningkatan kecepatan proses (*speed up*) yang menurun dapat disebabkan beberapa hal, seperti *latency* dari infrastruktur jaringan (komunikasi data), algoritma paralelisasi yang kurang sesuai dengan sistem komputer dan jaringan yang digunakan, jumlah proses yang terlalu banyak dibandingkan ketersediaan *core utility* yang dapat diberikan oleh sistem, dsb.

Hasil yang didapatkan dari pengujian-pengujian yang telah dilakukan ada yang menunjukkan terjadinya percepatan untuk semua jumlah proses, terjadi percepatan untuk sejumlah proses dan perlambatan untuk sejumlah lainnya, sampai dengan terjadi perlambatan pada semua jumlah proses yang diujikan. Berikut ini rangkuman dari hasil pengujian dan pembahasan yang telah dilakukan:

1. Pelacakan Optimal dengan kedalaman 22-

Tabel 4. Ringkasan Proses Tercepat Pelacakan A* 22 Level

No	Jml Pros	No. Pros	Level	Parent Nodes	Nodes	Total Nodes	Parent Elapse	Elapse	Total Elaps	Speed Up	S.U. Rel	Spdup/ Proses
1	**1**	0	22	0	3129	**3129**	0	46	**46**	1	**-**	1
2	**2**	0	22	0	2449	**2449**	0	140	**140**	0,3286	0,3286	0,16429
3	**4**	0	22	0	1670	**1670**	0	171	**171**	0,2690	0,8187	0,06725
4	**8**	0	22	0	2317	**2317**	0	234	**234**	0,1966	0,7308	0,02457
5	**16**	0	22	0	1605	**1605**	0	546	**546**	0,0842	0,4286	0,00527
6	**32**	0	26	0	2222	**2222**	0	812	**812**	0,0567	0,6724	0,00177
7	**64**	0	26	0	1461	**1461**	0	1468	**1468**	0,0313	0,5531	0,00049
8	**128**	0	26	0	1218	**1218**	0	3390	**3390**	0,0137	0,4330	0,00011

level terjadi percepatan untuk semua jumlah proses mulai 2, 4, 8, s.d. 128-proses, dan percepatan tertinggi pada 16-proses dengan percepatan sekitar 360 kali dibandingkan kecepatan (*throughput*) 1-proses.
2. Pelacakan Optimal 18-level terjadi percepatan sekitar 7 s.d. 8 kali untuk jumlah proses 2, 4, 8, 16-proses, 3 kali pada 32-proses, 2 kali pada 64-proses, tetapi untuk jumlah 128-proses terjadi perlambatan (0,5 kali *throughput* 1-proses).
3. Pelacakan Optimal 9-level, terjadi perlambatan untuk semua jumlah proses yang diujikan, mulai dari 2, 4, 8, s.d. 128-proses.
4. Pelacakan menggunakan Algoritma A*, terjadi perlambatan untuk semua jumlah proses yang diujikan, mulai dari 2, 4, 8, s.d. 128-proses.

Dalam pengujian yang telah dilakukan, didapatkan beberapa percobaan dengan hasil seperti pelacakan "tanpa solusi". Penulis juga sebenarnya melakukan percobaan untuk algoritma *Best First Search* yang menghasilkan suatu kondisi terjebak *looping node* yang pernah diekspansi karena representasi masalah yang digunakan adalah *Tree*.

Untuk mendapat pemahaman yang lebih mengenai proses paralelisasi, program dapat diujikan pada *platform* atau sistem komputer yang beragam, misalkan:

1. Menggunakan perangkat keras dan sistem operasi yang sama, tetapi dengan jumlah *memory* utama (RAM) yang berbeda-beda.
2. Menggunakan komputer dengan jumlah *core*, *core speed*, dan/atau arsitektur *core* yang berbeda.
3. Menggunakan sistem operasi yang berbeda dan/atau *library* pendukung paralelisasi yang lain.
4. Menggunakan sistem komputer paralel, HPC, komputasi *Grid*, dan/atau jaringan komputer terdistribusi yang lainnya.

Selain hal-hal tersebut di atas, untuk mendapatkan pemahaman lebih mengenai proses paralelisasi kita juga dapat menulis ulang algoritma paralel. Demikianlah hal-hal yang dapat kami sampaikan mengenai paralelisasi proses secara umum dan khususnya mengenai algoritma pelacakan A* dan Optimal (Dijkstraa) dalam ruang masalah *Tree* untuk menyelesaikan masalah *Puzzle* Kotak-8. Semoga bermanfa'at.

DAFTAR PUSTAKA

[1] Agust Isa Martinus. "Pelacakan Optimal Dalam Tree Secara Paralel Menggunakan Message Passing Pada Masalah Puzzle Kotak–8," dalam *Prosiding Festival & Seminar Tahunan Seni & Peradaban Tingkat Internasional*, 2015, pp. 319-329.

[2] Agust Isa Martinus. "Pemrosesan Paralel Menggunakan Message Passing untuk Pelacakan dengan Algoritma A* dan Optimal Pada Masalah Puzzle Kotak–8 Dalam Ruang Masalah Tree," dalam *Prosiding Seminar Nasional Teknik 2015 Fakultas Teknik Universitas Muhammadiyah Purwokerto*, 2015, pp. 213-222.

[3] Ananth Grama, Anshul Gupta, George Karypis, dan Vipin Kumar. 2003. *Introduction to Parallel Computing*. Addison Wesley, USA. 2nd Edition.

[4] Michael J. Quinn. 2003. *Parallel Programming in C with MPI and OpenMP*. McGraww-Hill Education, New York, NY – USA. 2nd Edition.

[5] Stuart Russel & Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education, Upper Saddle River, NJ – USA. 2nd Edition.

[6] Ted G. Lewis dan Hesham El-Rewini. 1992. *Introduction to Parallel Computing*. Prentice-Hall Interbational Inc., Englewood Cliffs, New Jersey – USA. 2nd Edition.

TENTANG PENULIS

Penulis, Agust Isa Martinus, adalah pengajar pada Program Studi Teknik Informatika, Fakultas Teknik – Universitas Muhammadiyah Cirebon, tinggal di kota Cirebon. Latar belakang pendidikan penulis adalah sarjana dan magister dalam bidang Teknik Komputer sehingga penulis belajar mengenai perangkat keras komputer secara sistem maupun perancangan microprocessor, sistem operasi tingkat rendah (low level, yang berhubungan dengan perangkat keras), bahasa pemrograman komputer tingkat rendah dan menengah, dan algoritma-algoritma komputasi. Penulis bergabung sebagai pengajar UMC (Universitas Muhammadiyah Cirebon) pada Program Studi Teknik Informatika dari tahun 2005 sampai dengan sekarang.