E-ISSN : 2722-0346

Parallel Processing Pada Pemodelan *Machine Learning* Menggunakan *Random Forest*

N.I.S. Baldanullah[-1], Naufal Mulyarizki[-2], Indah Permatasari[-3], Iqbal Putra Naufal[-4]
Dimas Candra Pratama[-5]
Program Studi Teknik Informatika[1,2,3,4,5]
Fakultas Teknik Universitas Pancasila Jakarta[1,2,3,4,5]
baldanullah55@gmail.com[1], naufalmulyarizki09@gmail.com[-2], indahp3101@gmail.com[-3],
iqbalputranaufal04@gmail.com[-4], dimasunknown912@gmail.com[-5]

Abstrak*—Algoritma *Random Forest* dalam melakukan pengklasifikasian dengan membuat beberapa *decision tree* pada setiap sampel yang dipilih kemudian membuat hasil prediksi dari setiap *decision tree* dan memilih hasil prediksi akhir berdasarkan *vote* terbanyak. Penelitian ini akan membandingkan *execution time* yang diproses menggunakan teknik *serial processing* dengan *parallel processing* pada saat melakukan *training* data dengan menggunakan dataset *flight delay* dengan jumlah baris data sebanyak 563.737 baris. Hasilnya menunjukkan bahwa secara rata rata *parallel processing* mampu memproses *training* data lebih cepat berdasarkan tiap jumlah *n* buah *decision tree* yang telah ditentukan. Namun, jika membandingkan *core* pada *parallel processing* itu sendiri, seperti percobaan dengan jumlah 10 dan 20 buah *decision tree, execution time* dihasilkan lebih cepat dengan menggunakan 6 dan 7 *core* daripada menggunakan 8 *core.* Hasil akurasi terbesar didapatkan dengan penentuan jumlah buah *decision tree* sebanyak 40 dan 50 buah dengan akurasi yang sama sebesar 78.14%.

Kata Kunci — Parallel Processing, Random Forest, Machine Learning

I. PENDAHULUAN

Pemrosean paralel adalah proses komputasi dengan lebih dari satu tugas pada suatu waktu secara bersamaan yang bertujuan mempersingkat waktu dalam penyelesaian *tasks* dengan memanfaatkan serta mengoptimalkan sumber daya pada komputer yang ada [1]. Perkembangan teknologi *multicore* menjadikan teknik proses paralel menjadi berkembang pesat dengan adanya lebih dari satu *core* pada *processor*, sehingga *processor* mampu melakukan komputasi lebih dari satu yang terpisah pada waktu yang bersamaan [2].

Penerapan *machine learning* memerlukan berbagai tahapan yang dilalui seperti mengidentifikasi data, mempersiapkan data, memilih algoritma *machine learing*, membagi data, melatih data kemudian mengevaluasi model, serta uji coba model. Dari tahapan pemodelan mac*hine learning* tersebut, tentu kecepatan dan ketepatan sangat dibutuhkan agar tidak memakan waktu yang banyak.

Terdapat banyak algoritma yang beragam, beberapa metode *machine learning* seperti menggunakan model *ensemble learning*. Pada pemodelan machine learning ini menggunakan kombinasi dari beberapa algoritma untuk mendapatkan akurasi yang baik, salah satunya dengan menggunakan metode *boostrap sampling* pada algoritma *Random Forest* dimana alogitma ini membuat lebih dari satu *decision tree* kemudian algoritma ini melakukan *voting* dari *tree* terbaik yang terbentuk.

Random Forest digunakan dalam pengklasifikasian data set dalam jumlah yang besar, karena memiliki fungsi yang bisa digunakan pada banyak dimensi dengan berbagai skala [3]. Sehingga akan memerlukan waktu pada proses training data pada modelnya.

Strategi dalam pemrosesan paralel memerlukan dukungan *hardware* namun belum cukup dalam pengembangan strategi paralel, sehingga diperlukan dukungan *software* serta *library* pada bahasa pemrograman seperti *python* yang mendukung kemampuan paralel serta pendukung lainnya yang mampu digunakan dalam pemodelan *machine learning* seperti *jupyter notebook* serta *library Scikit Learn* dengan berbagai *module* dalam pemodelan *machine learning.*

Pada penelitian ini, akan dilakukan perbandingan *execution time* melalui *parallel processing* dengan 2 - 8 *core* dan juga *serial processing* pada pemodelan *machine learning* menggunakan algoritma *Random Forest* pada tahap data *training*.

Penelitian ini menggunakan *Dataset* dari *website kaggle* yang besumber dari situs resmi *Bureau of Transportaion Statistic* berupa data *flight delay* yaitu data mengenai status keterlambatan pesawat pada riwayat penerbangan di Amerika Serikat. *Dataset* ini akan di *input* pada media pemodelan *machine learning* yaitu *jupyter notebook* sebagai data dalam pemodelan *machine learning* dengan algoritma *random forest* menggunakan *library Sklearn* pada *python.*

Banyak penelitian yang telah dilakukan dalam membandinkan *execution time* pada *serial processing* dengan *parallel processing*, seperti pada penelitian [4], penelitian tersebut melakukan perbandingan *execution time* antara *serial* dan *parallel processing* dalam klasterisai menggunakan algoritma *K-Means* dengan hasil bahwa *parallel processing* mampu mengurangi *execution time* hingga 52%. Penelitan [5] juga menghasilkan kesimpulan bahwa *parallel processing* lebih cepat 4 kali lipat daripada *serial processing* pada proses visualisasi *geospatial.*

Namun, pada penelitian [6] menyimpulkan bahwa secara umum untuk jumlah data yang tidak terlalu besar, waktu komputasi serial berjalan lebih cepat bila dibandingkan dengan waktu komputasi secara paralel, apabila jumlah data

yang diproses terbilang kecil sebaiknya hanya menggunakan dua prosesor saja dengan menggunakan komputasi secara serial. Namun, jika data sudah memasuki jutaan sebaiknya menggunakan algoritma *Radix Sort* dengan empat *processor* secara paralel. Dan apabila data yang diproses sudah sangat besar seperti puluhan juta, sebaiknya menggunakan algoritma *Radix Sort* dengan 3 komputer atau lebih secara paralel. Sehingga dapat dibuat hipotesis terkait penelitian tersebut bahwa *Parallel processing* tidak selalu menghasilkan *execution time* lebih cepat dibanding dengan *serial processing* tergantung pada ukuran data serta algoritma yang digunakan.

Penelitian ini bertujuan untuk melakukan perbandingan *execution time* terhadap tahapan *training* data pada algoritma *machine learning* menggunakan algoritma *Random Forest* dalam melakukan prediksi serta menghitung optimasi yang dihasilkan terhadap masing masing teknik *processing.*

II. METODE PENELITIAN

Penelitian ini dilakukan dengan mengimplementasikan model *machine learning* menggunakan algoritma *Random Forest* dalam melakukan prediksi pada *tools jupyter notebook* dengan memanfaatkan *library SKLearn* dalam pemodelan *machine learning* dan *module joblib* pada *python* dalam melakukan *parallel processing*.

Berikut merupakan *gambar* 1 berupa *Flowchart* dari tahapan metode penelitian.

A. Perolehan Dataset

Penelitian ini menggunakan dataset berupa *file* (.csv) yaitu data mengenai riwayat penerbangan pesawat di negara Amerika Sertikat diambil dari *website kaggle* dimana *dataset* ini bersifat publik dengan lisensi *CC0* sehingga data ini dapat disalin, dimodifikasi, serta didistribusikan tanpa meminta izin.

Dataset memiliki total baris sebanyak 563.737 baris serta 120 kolom, *dataset* tersebut akan di *input* ke *tools jupyter notebook* kemudian dilakukan pengolahan data untuk digunakan dalam pemodelan *machine learning*.

Gambar.1 Flowchart Metode Penelitian

B. Praproses

Tahapan ini melakukan praproses terhadap *dataset* yang ada sebelum melakukan pemodelan dengan tujuan untuk mempersiapkan dan memperoleh data yang sesuai dengan model *machine learning* seperti menghilangkan fitur fitur yang tidak diperlukan serta membagi data latih dan data uji.

Pada tahapan ini, praproses dilakukan dengan teknik berikut antara lain:

1. Memilih fitur dan membuang fitur
   Dari perolehan dataset, diketahui terdapat 120 fitur pada data tersebut dimana sebelumnya pada pemodelan ini hanya akan digunakan 7 fitur utama sebagai bahan untuk pemodelan algoritma *machine learning*, sehingga dilakukan penghapusan fitur yang tidak digunakan dengan menggunakan *library pandas* pada *python*.
2. Mengisi *missing value*
   Dari dataset, terdapat *missing value* sebesar 0.06 % pada fitur target yaitu *ArrDel155* sehingga dalam mengatasi *missing value* tersebut dilakukan pengisian nilai dengan nilai 1 atau benilai *true* pada fitur tersebut.
3. Konversi data *categorical* ke *integer*
   Berdasarkan dataset yang ada, data yang akan menjadi data *feature* yang bertipe *categorical* akan diubah menjadi tipe *integer*. Dalam pemodelan *machine learning*, sebagian besar input datanya harus berbentuk numerikal. Maka dari itu data yang berupa kategorikal tersebut harus diubah terlebih dahulu menjadi data numerikal agar model *machine learning* bisa memproses data tersebut [7]. Pada tahapan ini dilakukan dengan menggunakan teknik *encoding*.
4. Memisahkan data *features* dan data *target*
   Tahapan ini melakukan pemisahan dengan membuat dua variabel yaitu *X* dan *y* serta menginisialisasikan nilai *X* sebagai data *features* dan y sebagai data *target*.
5. Membagi data latih dan data uji
   Tahapan ini data dibagi dengan perbandingan 80:20 dimana data latih sebesar 80% dan data uji sebesar 20%.
   Data latih akan digunakan sebagai data sebagai bahan dalam *machine learning* menggunakan algoritma *Random Forest*, sedangkan data uji digunakan dalam menguji performa model yang didapatkan.

C. Membangun model machine learning

1. *Tools, Package dan library*
   Pemodelan *machine learning* pada penelitian ini dengan memanfaatkan *tools, package* dan *library* yang sudah ada yaitu:

   Python
   Bahasa pemrograman ini digunakan secara luas dalam melakukan pengembangan aplikasi karena sifatnya yang interpretatif dengan *syntax* yang sintuirif serta memiliki beragam *package* dan *library* yang mendukung pengembangan *data science*, *machine learning* serta *IoT* [12].

   Jupyter Notebook
   Berupa *tools* yang bersifat *opensource* yang berfungsi sebagai media dalam pemodelan *machine learning*,

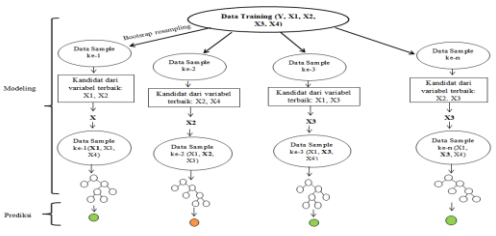
fokus dari *tools* ini sebagai alat untuk menganalisis serta eksperimen data yang cepat dan handal [11].

ScikitLearn
Library ini mebantu dalam melakukan berbagai tahapan pada pemodelan *machine learning* dengan beragam *module* dan *package* didalamnya yang digunakan untuk pemodelan seperti klasifikasi maupun regresi [13].

Joblib
Module ini memiliki sekumpulan kode yang mampu melakukan *parallel processing*, sehingga *library* yang terdapat akan dipanggil dan dieksekusi secara bersamaan pada tahapan *training data*.

Python Time Module
Module ini mampu melakukan proses perhitungan waktu pada tahapan proses yang dilakukan, sehingga hasil pengukuran *execution time* akan diukur menggunakan *library* tersebut.

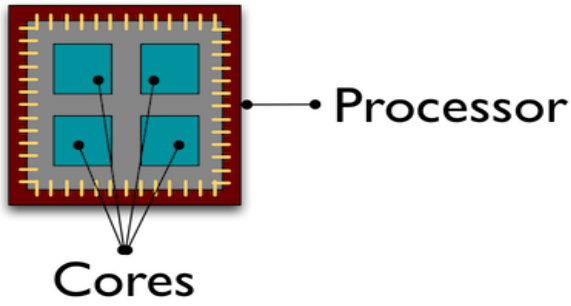
2. *Random Forest*
Random forest merupakan metode *machine learning* yang bersifat supervised dalam pengklasifikasian, dengan mengkombinasikan lebih dari satu prediksi dari *multiple decision tree* [8].
Metode ini merupakan sebuah ensemble (kumpulan) metode pembelajaran menggunakan pohon keputusan sebagai base classifier yang dibangun dan dikombinasikan *decision tree* atau pohon pengambil keputusan adalah sebuah diagram alir yang berbentuk seperti pohon yang memiliki sebuah *root node* yang digunakan untuk mengumpulkan data [9].
Ada tiga aspek penting dalam metode random forest, yaitu: (1) Mengambil data *sample* sebanyak *n* kali; (2) Dari setiap data *sample* kemudian memilih kandidat variabel input terbaik; (3) Variabel input yang tidak dijadikan kandidat digunakan dalam membentuk model bersama variabel input terbaik; (4) Pengambuilan keputusan untuk prediksi data didasarkan pada prinsip *aggregating* dengan *voting* dari hasil prediksi keseluruhan model [10].
Dalam menerapkan algoritma *Random Forest* pada penelitian ini pemodelan dilakukan menggunakan fungsi *RandomForestClassifier()* pada *library SciKit Learn.*
Berikut merupakan *Gambar2* berupa ilustrasi pada algoritma *Random Forest*



Gambar.2 Ilustrasi algoritma *random forest* [10]

3. *Parallel Processing*
Processor merupakan sebuah komponen dari sebuah komputer yang berfungsi untuk mengontrol segala proses yang berjalan pada suatu perkangkat komputer serta mampu melakukan komputasi dan menjalankan tugas. Perkembangan teknologi processor terus berkembang yang pada awalnya hanya memiliki *single core* kini *processor* memiliki lebih dari satu *core* yang disebut sebagai *multi-core*. Penambahan *core* ini menjadikan komputer mampu melakukan lebih dari satu *task* dalam waktu yang sama, dengan demikian performansi dari komputer dapat meningkat. Suatu komputer yang memiliki satu *prosesor* yang terdiri atas 4 core dapat melakukan 4 komputasi dalam waktu bersamaan. Keempat *core* tersebut terkoneksi dengan *main memory* yang sama [2]. *Gambar 3* berikut merupakan ilustrasi *processor*.

Core merupakan komponen fisik pada komputer sedangkan istilah *thread* merupakan komponen virtual, *core* mampu meningkatkan jumlah *task* yang di eksekusi dalam suatu waktu dan *thread* mengatur serta mengoptimalkan beban pada *task* tersebut.
Kinerja suatu proses pada komputer bergantung pada jumlah *core* serta kecepatan *core* tersebut dalam melakukan eksekusi suatu instruksi. Pada pemrosesan serial, suatu *thread* dapat dieksekusi setelah *thread* sebelumnya selesai dijalankan dan telah memperoleh hasil (*return*). Konsep dasar dari implementasi paralel yang membedakan dengan implementasi secara serial adalah *thread* pada paralel dapat dijalankan secara



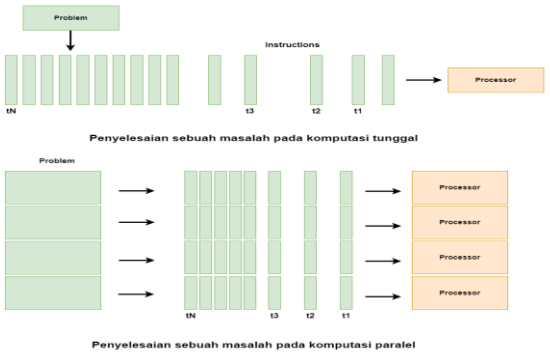
Gambar.3 Ilustrasi model *processor* 4 *cores* [2]

bersamaan pada satu waktu. Oleh karena itu, data yang digunakan pada tiap *thread* harus tidak saling berkaitan satu sama lain sehingga dapat diproses secara terpisah. Apabila data yang digunakan merupakan output dari proses sebelumnya maka kedua proses ini tidak dapat dijalankan secara paralel [2].

 E-ISSN : 2722-0346

Berikut merupakan *Gambar 4* berupa skema *serial processing* dan *multi processing*.

4. *Training* data
Data *training* merupakan kumpulan data yang akan digunakan sebagai bahan dalam melatih suatu



Gambar.4 Ilustrasi *single* dan *parallel processing*

algoritma pada *machine learning*, pada penelitian ini dataset yang sudah ada digunakan untuk melakukan *training* pada algoritma *machine learning* yaitu algoritma *Random Forest* melalui proses *serial processing* dan *parallel processing.*

D. Evaluasi hasil perbandingan

Tahapan ini dilakukan dengan mengakumulasi hasil *execution* time proses *training* kemudian hasilnya di tabulasi pada sebuah tabel dengan menghitung rata rata hasil *execution time* pada n buah *decision tree* sebanyak 6 kali percobaan menggunakan *serial processing* dan *parallel processing* dengan 2 sampai 8 *core*.

E. Pengujian Akurasi Model

Pengukuran akurasi pada model yang ada dilakukan dengan menggunakan metode *confusional matrix* dimana metode ini dilakukan untuk mengukur performa dalam suatu prediksi model *machine learning* terhadap pengklasifikasian yang memiliki minimal dua label atau *class*.

III. HASIL DAN PEMBAHASAN

Pengaruh *serial processing* dengan *parallel processing* terhadap *execution time* pada *training data* terhadap pemodelan *machine learning* menggunakan algoritma *Random Forest* perbandingannya diuji dengan menggunakan *tools jupyter notebook* serta *library* pada bahsa pemrograman *python* dengan *package* dan *module* di dalamnya yang mendukung dalam pemodelan *machine learning* dan pemrosesannya secara paralel.

Data yang digunakan pada penelitian ini merupakan data dengan ukuran yang teregolong besar dengan total baris sebanyak 563.737 baris dengan total kolom sebanyak 120, kemudian dilakukan praproses data dengan menghilangkan fitur pada data yang tidak diperlukan sehingga dataset yang digunakan dalam *training* hanya menggunakan 7 kolom dengan menargetkan fitur *ArrDel15* yaitu data indikator terkait data kedatangan pesawat dengan keterlambatan 15 menit atau lebih dengan nilai *boolean* berupa 1 dan 0 dimana angka 1 mengindikasikan bahwa pesawat tersebut terlambat dan angka 0 mengindikasikan bahwa pesawat tersebut tidak terlambat lebih dari 15 menit.

Dalam melakukan pemodelan *machine learning* perlu dukungan perangkat *hardware* dan *software yang* mumpuni sehingga proses pemodelan bisa dilakukan secara efisien. Berikut merupakan *Tabel I* yaitu spesifikasi *hardware* dan *software* yang digunakan dalam penelitian ini.

Pengujian perbandingan *execution time* dilakukan berdasarkan penentuan jumlah *decision tree* pada fungsi `RandomForestClassifier()`. Penentuan *decision tree* ini dengan menentukan sejumlah *n* buah *decision tree* pada

TABEL I
SPESIFIKASI HARDWARE DAN SOFTWARE

No	*Hardware/software*	Spesifikasi
1	*Processor*	*11th Gen Intel(R) Core (TM) i7-11800H @ 2.30GHz (16 CPUs), ~2.3GHz*
2	Jumlah *Core*	8 *Core*
3	*RAM*	16 *Giga byte*
4	*Operating System*	Windows 11
5	*Jupyter Notebook*	*Version* 6.5.2

input parameter yaitu `n_estimators` pada fungsi `RandomForestClassifier()` dari *library Sklearn*.

Pada pemodelan *machine learning* dengan menggunakan Algoritma *Random Forest* akan membuat sejumlah *n* buah *decision tree* pada tahapannya, penelitian ini akan melakukan perbandingan *execution time* dengan berbagai penentuan *n* buah pada algoritma tersebut menggunakan *serial proscessing* dan *parallel processing*. Tahapan ini melakukan training data menggunakan fungsi `RandomForestClassifier()` pada *Sklearn* dengan dua teknik pemrosesan dengan menentukan parameter berupa `n_estimators` pada fungsi `RandomForestClassifier()` dari *package Sklearn* yaitu jumlah *decision tree* yang akan dibangun oleh algoritma *Random Forest* sehingga proses *training* dengan n jumlah *decision tree* ini akan dibandingkan berapa lama waktu yang dihasilkan pada saat melakukan proses *training* dengan n buah *decision tree* menggunakan kedua teknik *processing* sebanyak 6 kali percobaan.

Fungsi pada *package Sklearn* yaitu `RandomForestClassifier()` memiliki parameter yang terintegrasi dengan *module* pada *python* berupa *joblib* dimana modul ini memberikan fungsionalitas dalam melakukan pemrograman paralel. *Joblib* memiliki parameter berupa `n_jobs` yang menjadi parameter pada fungsi `RandomForestClassifier()`, parameter ini akan di *input* dengan sebuah angka *integer* yang mewakili jumlah c*ore* yang akan digunakan dalam melakukan proses *training*.

Berikut merupakan *Tabel II* berupa *n* buah *decision tree* serta penggunaan *core* yang dipakai dalam memproses algoritma *Random Forest* dengan masing masing *n.*

TABEL II
JUMLAH *CORE* DAN *DECISION TREE*

core	*n decision tree*
1, 2, 3, 4, 5, 6, 7, 8	5 10 20 30 40 50

n	Percobaan					*Mean*
	1	2	3	4	5	
5	26,56	27,01	27,10	26,80	26,62	26,82
10	53,47	53,66	53,46	53,96	53,70	53,65
20	102,20	103,00	103,47	103,16	102,45	102,86
30	156,77	154,77	156,54	156,71	155,56	156,07
40	201,84	202,35	201,97	201,87	201,78	201,96
50	259,95	257,37	258,66	259,11	259,55	258,93

Gambar.5 Hasil *execution time* menggunakan *single core*

n	Percobaan					*Mean*
	1	2	3	4	5	
5	18,36	18,38	18,36	18,26	18,35	18,34
10	32,96	32,88	32,84	32,76	32,08	32,70
20	62,78	62,37	62,39	62,88	62,10	62,50
30	93,42	94,20	93,38	93,78	94,23	93,80
40	123,68	123,30	123,65	123,68	123,12	123,49
50	155,85	153,99	154,67	153,51	153,49	154,30

Gambar.6 Hasil *execution time* menggunakan *2 core*

Pengukuran *execution time* dilakukan dengan memanfaatkan *module time* pada *python*. Fungsi `time.time()` pada *module time* berfungsi dalam melakukan perhitungan waktu pada suatu proses di dalam *cell* tertentu, dalam hal ini fungsi tersebut akan menghitung waktu *execution time* pada saat melakukan *training data* menggunakan algoritma *Random Forest.*

Fungsi tersebut dipanggil sebanyak dua kali dalam satu pengukuran proses, dimana perintah pertama melakukan *start time* dan perintah kedua melakukan *end time.* Hasil total *execution time* akan diukur dengan mengurangi *end time* dan *start time*. Berikut merupakan kode program dalam melakukan pengukuran *execution time* pada tahap *training* model.

Berikut merupakan kode program bahasa *python* pada *jupyrter notebook* pada tahapan *training model* menggunakan *library SciKit Learn* serta *module* yang terdapat pada *python.*

```
from sklearn.ensemble import
RandomForestClassifier
import time
model_list =
RandomForestClassifier(n_estimators=n, n_jobs=n,
random_state=0)
tic = time.time()
model_list.fit(X_train, y_train)
toc = time.time()
print(f" {toc-tic} seconds")
```

Berdasarkan tahapan yang dilakukan, setelah melakukan proses pengujian dengan menggunakan skenario yaitu menguji *execution time* menggunakan 1 *core* pada *processor* dalam mengeksekusi *task* yang disebut sebagai *serial processing* serta 2 sampai 8 *core* yang disebut sebagai *multi processing* terhadap *n* jumlah *decision tree* yang ditentukan yaitu 5, 10, 20, 30, 40 dan 50 jumlah *decision tree*. Berikut merupakan hasil *execution time* yang dengan satuan detik.

Dari Gambar 5 diatas, terlihat bahwa *execution time* dengan jumlah *decision tree* yang lebih sedikit mendapatkan waktu yang lebih cepat, semakin banyak jumlah *decision tree* maka semakin banyak jumlah perhitungan yang diproses sehingga semakin besar waktu yang dibutuhkan.

Berikutnya merupakan hasil *execution time* menggunakan pemrosesan paralel dengan melibatkan 2 sampai 8 *core* pada prosesnya terlihat pada Gambar 6. Dengan menggunakan *parallel processing* dengan 2 *core* terlihat bahwa ada perbedaan hasil *execution time* jika dibandingkan dengan *serial processing*, pada percobaan menggunakan 5 *decision tree, parallel processing* menghasilkan *execution* time lebih cepat 8.48 detik daripada menggunakan *serial processing* begitupun dengan percobaan yang dilakukan menggunakan jumlah *decision tree* yang lebih banyak, hasil dari *parallel processing* terhadap rata rata hasil *mean* dari setiap perobaan pada semua n *decision tree* yang dilakukan mampu mengoptimasi *execution running* terhadap *serial processing* sebesar 39.38% lebih cepat.

Hasil *execution time* dengan menggunakan 3 *core* lebih cepat dibandingkan dengan menggunakan 2 *core* namun terlihat bahwa optimasinya semakin kecil berdasarkan rata rata hasil *mean* dari setiap percobaan pada semua n *decision tree* yang dilakukan mampu mengoptimasi *execution time* terhadap 2 *core* sebesar 18.97%.

Dari percobaan menggunakan 4 *core* terlihat bahwa pada saat pengujian dengan jumlah *decision tree* terkecil hasil

n	Percobaan					*Mean*
	1	2	3	4	5	
5	14,62	14,67	14,70	14,53	14,62	14,63
10	28,02	27,93	27,84	27,91	27,58	27,85
20	50,47	50,80	51,03	50,84	50,38	50,70
30	74,20	74,74	73,77	74,40	74,12	74,25
40	99,15	101,82	105,23	102,31	101,30	101,96
50	123,18	124,87	124,14	122,82	123,56	123,72

Gambar.7 Hasil *execution time* menggunakan *3 core*

yang didapatkan pada percobaan pertama lebih lama dibandingkan dengan menggunakan 3 *core.* Selain itu, terlihat bahwa optimasinya juga semakin kecil berdasarkan rata rata hasil *mean* dari setiap percobaan pada semua *n decision tree* yang dilakukan dengan nilai optimasi sebesar 12.16% terhadap 3 *core*

n	Percobaan					*Mean*
	1	2	3	4	5	
5	14,64	14,55	14,48	14,51	14,49	14,53
10	24,46	24,97	24,71	24,66	24,89	24,74
20	44,51	44,42	44,28	44,46	44,45	44,42
30	66,81	66,75	66,49	66,63	66,69	66,67
40	86,90	86,42	86,17	86,26	86,89	86,53
50	108,39	108,20	108,47	108,23	108,87	108,43

Gambar.8 Hasil *execution time* menggunakan *4 core*

Berdasarkan pengujian dengan 1-4 *core* terlihat bahwa semakin banyak *core* yang digunakan semakin kecil selisih *execution time* yang dihasilkan dengan hasil *execution time* menggunakan *core* yang lebih sedikit.

Pada pengujian menggunakan 4 *core* diketahui bahwa terdapat hasil *execution time* yang lebih lama dibandingkan dengan 3 *core* dengan jumlah *decision tree* terkecil yaitu 5, pada pengujian menggunakan 5 *core* ini terlihat lebih cepat daripada menggunakan 3 dan 4 *core*. Namun, dengan jumlah *decision tree* sebanyak 10 hasil yang didapatkan pada percobaan pertama justru lebih lama dibandingkan dengan menggunakan 4 *core*, optimasinya juga semakin kecil berdassarkan rata rata hasil *mean* dari setiap percobaan pada

 E-ISSN : 2722-0346

n	Percobaan					Mean
	1	2	3	4	5	
5	11,61	11,39	11,50	11,28	11,38	11,43
10	24,66	24,76	24,61	24,52	24,64	24,64
20	40,96	40,88	41,24	41,02	40,90	41,00
30	61,51	61,27	61,29	61,29	61,14	61,30
40	79,63	80,27	80,00	80,92	82,11	80,59
50	100,06	100,10	101,65	101,98	102,16	101,19

Gambar.9 Hasil *execution time* menggunakan *5 core*

semua n *decision tree* yang dilakukan yaitu sebesar 7.29% terhadap 4 *core.*

n	Percobaan					Mean
	1	2	3	4	5	
5	11,45	11,27	11,44	11,55	11,40	11,42
10	21,65	21,65	21,62	21,52	21,37	21,56
20	41,07	41,06	41,33	41,14	41,30	41,18
30	58,43	58,36	58,39	58,36	58,65	58,44
40	76,79	77,69	77,72	77,08	77,71	77,40
50	96,93	96,46	96,89	96,30	96,78	96,67

Gambar.10 Hasil *execution time* menggunakan *6 core*

Selanjutnya pada Gambar 10 merupakan hasil *execution time* menggunakan 6 core dimana dalam menguji *execution time* dengan jumlah *decision tree* sebanyak 5 hasilnya bisa dibilang sama saat menggunakan 5 *core* dengan rata rata hasil *execution time* masing masing 11.42 detik dan 11.43 detik. Hasil *execution time* dengan jumlah *decision tree* sebanyak 20 terlihat bahwa dengan menggunakan 5 *core* justru lebih cepat daripada menggunakan 6 *core* meskipun secara optimasi *execution time* menggunakan 6 *core* juga semakin kecil berdasarkan rata rata hasil *mean* dari setiap percobaan pada semua n *decision tree* yang dilakukan sebesar 4,21% terhadap 5 *core*

n	Percobaan					Mean
	1	2	3	4	5	
5	11,56	11,52	11,27	11,33	11,30	11,40
10	21,69	21,62	21,66	21,58	21,62	21,63
20	38,83	38,75	38,37	38,50	38,53	38,60
30	58,53	58,96	58,75	58,79	59,12	58,83
40	75,51	75,48	75,06	74,74	75,01	75,16
50	94,73	94,12	94,47	94,21	95,01	94,51

Gambar.11 Tabel hasil *execution time* menggunakan *7 core*

Berdasarkan pengujian *execution time* menggunakan 7 *core* dengan jumlah *decision tree* sebanyak 5 hasilnya juga tidak terlalu jauh beda dibandingkan dengan menggunakan 5 dan 6 *core* dengan rata rata hasil *execution time* sebesar 11.40 detik. Dengan jumlah *decision tree* sebanyak 10 dan 30 *decision tree*, menggunakan 7 *core* tidak memberikan hasil *execution time* yang lebih cepat dibandingkan dengan menggunakan 6 *core*, namun secara optimasi *execution time* menggunakan 7 core juga semakin kecil berdasarkan rata rata hasil mean dari setiap percobaan pada semua n decision tree yang dilakukan sebesar 2,13% terhadap 6 core.

Pengujian terkahir dengan menggunakan 8 *core* memberikan hasil bahwa dengan jumlah *decision tree* sebanyak 5 hasilnya berada pada rata-rata 11.36 detik yang tidak terlalu jauh secara signifikan dibandingkan dengan menggunakan 5, 6 dan 7 *core* namun secara keseluruhan menggunakan 8 *core* dengan jumlah *decision tree* 5 memberikan hasil *execution time* yang paling cepat dibandingkan yang lainnya, akan tetapi menggunakan 8 *core* dengan jumlah *decision tree* sebanyak 10 memberikan hasil *execution time* yang paling lama dibandingkan dengan menggunakan 6 dan 7 *core.* optimasi *execution time* menggunakan 8 core juga semakin kecil berdasarkan rata rata hasil mean dari setiap percobaan pada semua n decision tree yang dilakukan sebesar 1,85% terhadap 7 core. Berikut merupakan *Gambar 12* berupa hasil *execution time* menggunakan *core* sebanyak 8.

n	Percobaan					Mean
	1	2	3	4	5	
5	11,36	11,46	11,26	11,43	11,31	11,36
10	21,95	22,00	21,94	21,89	21,88	21,93
20	38,67	38,91	38,89	38,69	39,00	38,83
30	56,48	56,54	56,39	56,68	56,60	56,54
40	73,39	73,18	72,99	73,14	73,39	73,22
50	92,75	93,12	92,11	93,00	92,44	92,68

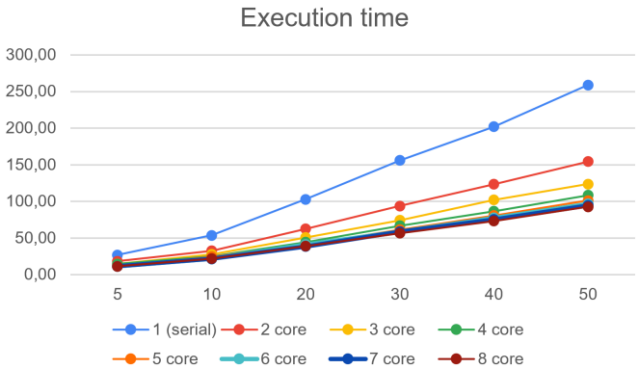
Gambar.12 Hasil *execution time* menggunakan *8 core*

Berikut merupakan rata rata *execution time* dengan 5 percobaan pada sejumlah *n* buah *decision tree* menggunakan teknik *serial processing* dan *parallel processing*.

n	Jumlah Core							
	1 (serial)	2	3	4	5	6	7	8
5	26,82	18,34	14,63	14,53	11,43	11,42	11,40	11,36
10	53,65	32,70	27,85	24,74	24,64	21,56	21,63	21,93
20	102,86	62,50	50,70	44,42	41,00	41,18	38,60	38,83
30	156,07	93,80	74,25	66,67	61,30	58,44	58,83	56,54
40	201,96	123,49	101,96	86,53	80,59	77,40	75,16	73,22
50	258,93	154,30	123,72	108,43	101,19	96,67	94,51	92,68

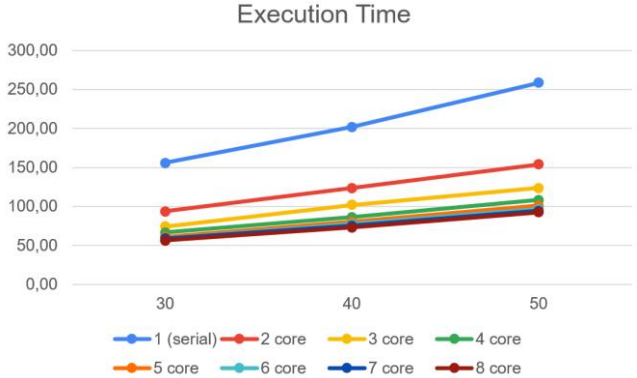
Gambar.13 Hasil perbandingan rata rata *execution time* menggunakan *serial processing* dan *parallel processing*

Berikut merupakan grafik perbandingan *execution time* menggunakan *serial processing* dan *parallel processing* pada sejumlah *n* buah *decision tree*.



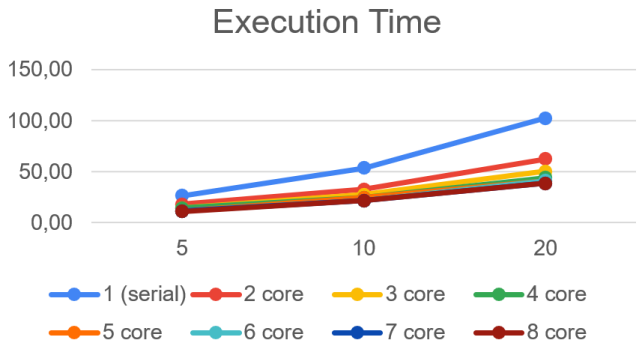
Gambar.14 Grafik hasil perbandingan *execution time* menggunakan *serial processing* dan *parallel processing*

Berdasarkan tabel dan grafik di atas, dapat dilihat bahwa perbandingan *execution time* menggunakan seluruh jumlah *core* secara rata rata memiliki hasil yang bervariasi perbedaan jarak optimasi *execution time* menggunakan *core* yang lebih tinggi terhadap *core* yang lebih rendah hasil perbedaannya terlihat jelas pada Gambar 15 percobaan menggunakan 30 sampai 50 jumlah *decision tree*, sedangkan dengan menggunakan 5 – 20 *decision tree* perbedaannya tidak terlalu signifikan dan beberapa percobaan hasil *execution time* nya menghasilkan nilai yang berada saling mendekati.



Gambar.15 Grafik hasil perbandingan *execution time* menggunakan *serial processing* dan *parallel processing* dengan *decision tree* sebanyak 30 – 50

Agar terlihat lebih jelas berikut merupakan grafik hasil *execution time* dengan sejumlah 30 - 50 buah *decision tree* menggunakan *serial processing* dan *parallel processing.*



Gambar.16 Grafik hasil perbandingan *execution time* dengan *decision tree* sebanyak 30 – 50

Gambar 16 Terlihat bahwa hasil perbandingan *execution time* menggunakan seluruh jumlah *core* dengan menggunakan *decision tree* sebanyak 30 – 50 menghasilkan grafik yang konstan pada setiap pengujian *core* dimana nilai hasil *execution time* tiap *core*nya tidak saling memotong dengan nilai hasil *execution time* pada *core* yang lain.

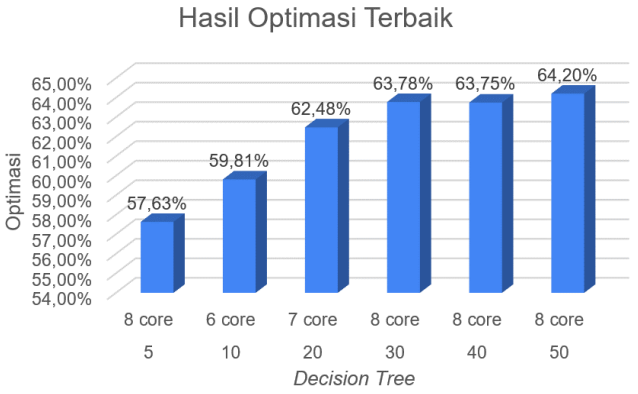
Sedangkan pada percobaan menggunakan *decision tree* sebanyak 5 – 20 menghasilkan beberapa titik nilai yang saling berpotongan pada pengujian menggunakan menggunakan 5, 6, 7 dan 8 *core*, hipotesis yang diharapkan adalah *execution time* yang dilakukan oleh *core* yang lebih kecil cenderung menghasilkan waktu yang lebih lama, namun perpotongan titik nilai ini mengindikasikan bahwa ada pengujian yang menghasilkan *execution time* lebih cepat menggunakan *core* yang lebih kecil pada *parallel processing* seperti yang ditunjukkan pada Gambar 17.

	5	10	20	30	40	50
2 core	31,61%	39,04%	39,23%	39,90%	38,86%	40,41%
3 core	45,46%	48,08%	50,70%	52,43%	49,51%	52,22%
4 core	45,81%	53,89%	56,81%	57,28%	57,16%	58,12%
5 core	57,38%	54,08%	60,14%	60,72%	60,10%	60,92%
6 core	57,41%	59,81%	59,96%	62,56%	61,68%	62,66%
7 core	57,51%	59,67%	62,48%	62,31%	62,79%	63,50%
8 core	57,63%	59,12%	62,25%	63,78%	63,75%	64,20%

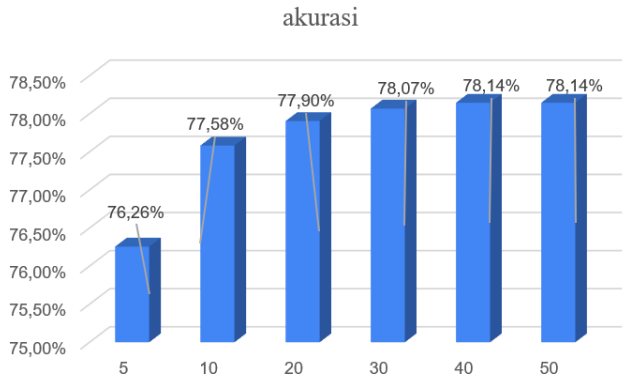
Gambar.17 Grafik hasil optimasi menggunakan *parallel processing* terhadap *serial processing*

Berdasarkan pengujian perbandingan *execution time* menggunakan *parallel processing*, berikut merupakan hasil persentase optimasi menggunakan *parallel processing* terhadap *serial processing*. Berikut merupakan Gambar 18 yaitu jumlah *core dengan* hasil optimasi *execution time* terbaik menggunakan *parallel processing* terhadap *serial processing* pada sejumlah *n* buah *decision tree*.

Dari sekian percobaan pengujian yang telah dilakukan pada tahapan *training*, berikut merupakan akurasi yang dihasilkan dengan menggunakan sejumlah *n* buah *decision tree* dapat dilihat pada Gambar 19.



Gambar.18 Grafik hasil optimasi menggunakan *parallel processing* terhadap *serial processing*



Gambar.19 Grafik hasil akurasi *Random Forest* terhadap *n* buah *decision tree*

IV. KESIMPULAN

Berdasarkan hasil dari percobaan mengukur *execution time* pada *training data* dengan algoritma *Random Forest* menggunakan teknik *serial processing* dan *parallel processing* bahwa secara rata rata *parallel processing* mampu memproses *training* tersebut lebih cepat berdasarkan tiap jumlah *n* buah *decision tree* yang telah ditentukan, pada penelitian ini menggunakan 2 sampai 8 *core* pada penerapan *parallel processing*. Hasilnya, seluruh hasil *execution time* mengguakan 2 sampai 8 *core* mampu mengoptimasi *execution time* terhadap *serial processing* yang menggunakan 1 *core*.

Namun, percobaan *parallel* processing dengan 2-8 *core* pada kasus ini tidak selalu memberikan hasil yang lebih cepat, jika membandingkan *core* pada *parallel processing* itu sendiri, seperti percobaan dengan jumlah 10 dan 20 buah *decision tree*, *execution time* dihasilkan lebih cepat dengan menggunakan 6 dan 7 *core* daripada menggunakan 8 *core.*

Berdasarkan pemodelan *machine learning* yang dilakukan dengan menggunakan algoritma *Random Forest*, akurasi terbesar dihasilkan dengan menggunakan 40 dan 50 buah *decision tree* dengan akurasi sebesar 78,14%.

DAFTAR PUSTAKA

[1] Mateti, Prabhaker, (2005), "Cluster Computing with Linux".
[2] B. Parhami, "Parallel Processing with Big Data," *Encycl. Big Data Technol.*, pp. 1–7, 2018. [online]. Available: https://web.ece.ucsb.edu/~parhami/pubs_folder/parh19b-ebdt-parallel-proc-big-data.pdf. [Accessed Nov. 15, 2022].

[3] Algorit.ma. (2022, 15 maret). *Cara kerja algoritma random forest*. [Online]. Available: https://algorit.ma/blog/cara-kerja-algoritma-random-forest-2022/. [Accessed Nov. 15, 2022].

[4] M. D. Marieska, dkk, "Optimasi algoritma k-mean clustering dengan parallel processing menggunakan framework r," *Jurnal Edukasi dan Penelitian Informatika*, vol.7, no.1, pp.70-75, 2021. [Online]. Available: https://jurnal.untan.ac.id/index.php/jepin/article/view/43400. [Accessed Nov.15, 2020].

[5] C. Ledur, D. Griebler, I. Manssour, and L. G. Fernandes, "A High-Level DSL for Geospatial Visualizations with Multi-core Parallelism Support," *Proc. - Int. Comput. Softw. Appl. Conf.*, vol. 1, pp. 298–304, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8029621. [Accessed Nov. 15, 2022].

[6] F. R. Lumbanraja, Aristoteles, N. R. Muttaqina, "Analisa komputasi paralel mengurutkan data dengan metode radix dan selection", *Jurnal Komputasi*, vol.8, no.2, pp.77-93, 2020. [Online] Available: https://jurnal.fmipa.unila.ac.id/komputasi/article/view/2662. [Accessed Nov. 15, 2022].

[7] A. Martulandi (2019, 14 Oktober), *Data Preprocessing pada Machine Learning: Handling Categorical Data #UcupStory*. [Online]. Available: https://medium.com/@adiptamartulandi/data-preprocessing-pada-machine-learning-handling-categorical-data-ucupstory-6e409dbfd0a0. [Accessed Nov. 15, 2022].

[8] V. Y. Kulkarni, P. K. Sinha, "Effective learning and classification using random forest algorithm", *International Journal of Engineering and Innovative Technology (IJEIT)*, vol3., no.11, pp. 267-273, Mei 2014. [Online]. Available: http://www.ijeit.com/Vol%203/Issue%2011/IJEIT1412201405_47.pdf. [Accessed Nov.15, 2022].

[9] M. G. Sadewo, A. P. Windarto, D. Hartama D, "Penerapan datamining pada populasi daging ayam RAS pedaging di indonesia berdasarkan provinsi menggunakan k-means clustering," *InfoTekJar (Jurnal Nasional Informatika dan Teknik Jaraingan)*, vol. 2, no.1, pp.60-67, 2017. [Online]. Available: https://jurnal.uisu.ac.id/index.php/infotekjar/article/view/164. [Accessed Nov. 15, 2022].

[10] S. Hanifah (2020, 13 Desember), *Credit Risk Analysis Menggunakan Random Forest*. [Online]. Available: https://silmihanifah1998.medium.com/credit-risk-analysis-menggunakan-random-forest-7a0a61ecb8ca. [Accessed Nov. 15, 2022]

[11] Algorit.ma. (2022, 15 maret). Cara menggunakan jupyter notebok. [online]. Available: https://algorit.ma/blog/cara-menggunakan-jupyter-notebook-2022/.

[12] M. Romzi, B. Kurniawan, "Pembelajaran pemrograman python dengan pendekatan logika algoritma," Jurnal Teknik Informatika Mahakarya JTIM, vol. 3, no. 2, pp. 37-44, Desember 2020. [online]. Available: https://journal.unmaha.ac.id/index.php/jtim/article/view/6. [Accessed Nov. 10, 2022]

[13] Bisa.AI Academy. (n.d.). *Machine Learning Dengan Scikit Learn Python. [online]. Available: https://www.bisa.ai/course/detail/MzU3/1*. [Accessed Nov. 10, 2022]