

# 大学院博士前期課程修士学位論文

## 題 目

OS仮想化基盤を用いたSBCマルチディスプレイシステムの  
フレーム処理並列化

指導教員

下條 真司 教授

報告者

高畠 勇我

2022年2月9日

大阪大学 大学院情報科学研究科

マルチメディア工学専攻

大学院博士前期課程修士学位論文

OS仮想化基盤を用いたSBCマルチディスプレイシステムのフレーム処理  
並列化

高畠 勇我

内容梗概

キーワード

## 目 次

1	序論	1
2	小型低性能計算機を用いたマルチディスプレイシステムの構築	2
2.1	従来のマルチディスプレイ構築手法 . . . . .	2
2.2	先行研究で提案されたマルチディスプレイシステム . . . . .	7
2.3	ヘッドノードの処理負荷軽減を行った関連研究 . . . . .	11
2.4	本研究での技術的課題 . . . . .	14
3	提案	15
3.1	OS仮想化基盤とコンテナ技術 . . . . .	15
3.2	MDのコンテナ化 . . . . .	15
3.3	コンテナを通したディスプレイの制御 . . . . .	15
3.4	ヘッドノードでのフレーム処理の改良 . . . . .	15
4	評価	17
4.1	実験環境 . . . . .	17
4.2	フレーム処理時間の評価 . . . . .	19
4.2.1	実験方法 . . . . .	19
4.2.2	実験結果 . . . . .	19
4.3	フレーム処理速度の評価 . . . . .	20
4.3.1	実験方法 . . . . .	20
4.3.2	実験結果 . . . . .	20
4.3.3	結果に対する考察 . . . . .	20
5	結論	21
	謝辞	22
	参考文献	23

# 1 序論

## 2 小型低性能計算機を用いたマルチディスプレイシステムの構築

本章では、小型低性能計算機、特に SBC を利用した低成本なマルチディスプレイシステム構築手法の必要性と、その実現に向けて行われた先行研究について説明する。まず 2.1 節では、従来利用されてきたマルチディスプレイシステムの構築手法とそのコストについて述べる。続く 2.2 節で先行研究でも使用されており、本研究でも取り扱うシングルボードコンピュータ (SBC) について説明し、その後 2.3 節で先行研究である、SBC を用いて構築されたマルチディスプレイシステムについて説明する。さらに 2.4 節で、先行研究で提案された SBC を用いたマルチディスプレイシステムについての問題点と、それを改善するための技術的な課題点について述べる。

### 2.1 従来のマルチディスプレイ構築手法

現在 MD の構築には数多くの手法が提案されており、それらは大きく分けると 2 種類の手法に分類することができる [1, 2]。1 つはマルチディスプレイを構築可能にする専用のハードウェアを用いて構築する手法、もう 1 つはマルチディスプレイを構築可能にする専用のミドルウェアを用いて構築する手法である。本節では、この 2 種類の構築手法を簡単に解説してそれぞれの構成例を示し、構築に要する金銭的コストについて説明する。

#### 専用ハードウェアを用いたマルチディスプレイの構築

マルチディスプレイ構築用のハードウェアは接続した複数台のディスプレイに対して映像出力用の信号を送信する機能を持った機器であり、主にデジタルサイネージ [3] や会議用のディスプレイを構築する際に利用される。マルチディスプレイ構築用ハードウェアの多くは、HDMI (High-Definition Multimedia Interface), DisplayPort [4] 等の規格に対応した映像出力ポートを複数搭載しており、各ポートに映像信号を分配することによって、接続したディスプレイを連動させて制御する。マルチディスプレイ構築用ハードウェアは、表示映像フレームの分割、接続ディスプレイ間の同期処理などを高速に行うことが可能であるため、高フレームレートでの映像出力が可能である。一方で、マルチディスプレイ構築用ハードウェアは入出力ポート数の上限が決まっているため、接続ディスプレイ数が制限される。そのため、マルチディスプレイを構築するディスプレイ数にも上限が存在することになり、スケーラビリティの観点からすると欠点となっている。マルチディスプレイ構築用ハードウェアの具体例としては、グラフィックボードやマトリックススイッチャが存在する。図 2.1 に、マトリックススイッチャを用いて構築したマルチディスプレイの概要図を示す。

グラフィックボードは、PC などのコンピュータにおいて、映像を信号として入出力するための機能を拡張ボードとして独立させたものである。各グラフィックボードに搭載されているチップやメモリによって、描画可能な解像度や表現色数、2D/3D 描画性能など



マトリックススイッチャ

図 2.1: システムの動作フロー

が異なる。マルチディスプレイを構築する目的でグラフィックボードを使用すると、ある程度の高解像な画像を描画可能であり、信号の入出力可能ポート数が多層備え付けられているものが必要となる。マルチディスプレイ対応型のグラフィックボードは、NVIDIA 社や AMD 社から発売されている。マルチディスプレイ対応型のグラフィックボードは、一般的に利用可能な映像出力ポート数が多く、表示解像度が優れているものほど価格が高くなる。

### 専用ミドルウェアを用いたマルチディスプレイの構築

マルチディスプレイ構築用ミドルウェアは、ディスプレイに接続した汎用 PC に画像フレームまたは描画命令を送信することによって動画や画像を表示する。マルチディスプレイ構築用ミドルウェアは主に科学的可視化の分野で広く利用されており、高性能なコンピュータで行われたシミュレーションの結果を表示する大規模可視化装置の構築などに使用される。接続可能なディスプレイ数に上限は設けられておらず、多数の汎用 PC を連携して動作させることで非常にスケーラブルなマルチディスプレイの構築が可能である。しかし、マルチディスプレイ構築用ミドルウェアは大規模な科学的可視化の用途を想定して設計されたものが多く、ディスプレイと接続された汎用 PC 群に対して高負荷な描画処理を要求する。そのため、マルチディスプレイを構築するために使用される汎用 PC はある程度高い性能を持ったものでなければならず、マルチディスプレイを構成するディスプレイ数が増加するごとに構築費用が高騰するという問題点がある。図 2.2 に、マルチディスプレイ構築用ミドルウェアを用いて汎用 PC 群を連携させることで構築したマルチディス

プレイの構成例を示す。

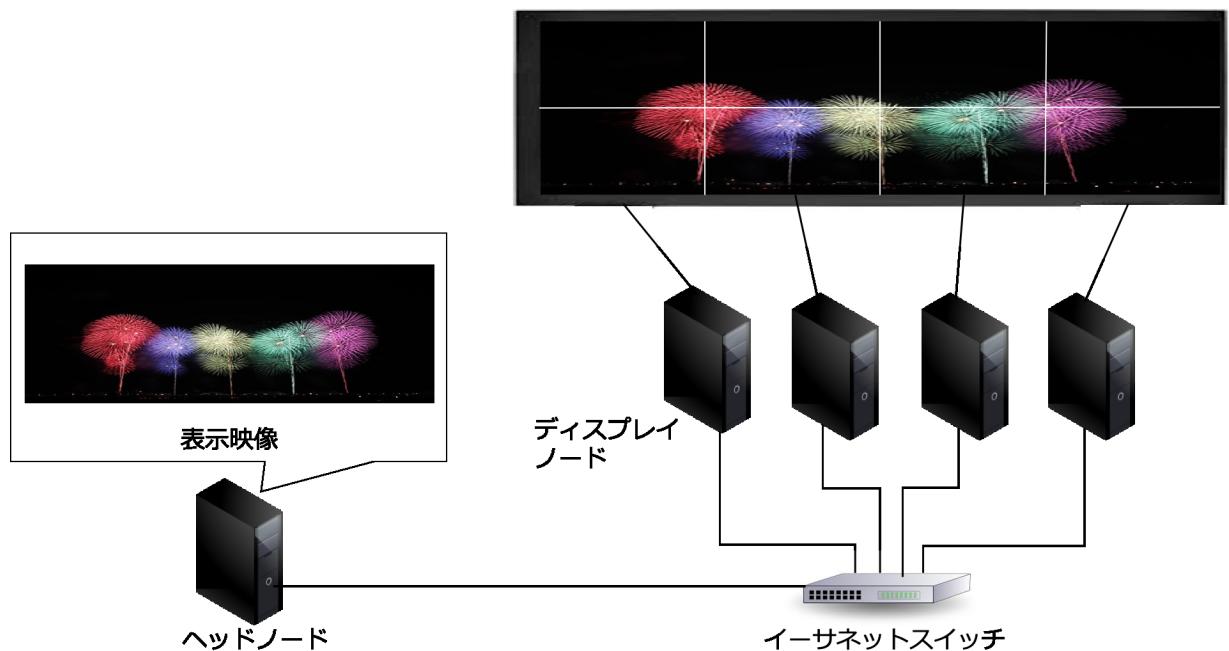


図 2.2: システムの動作フロー

それぞれの PC はマルチディスプレイ構築用ミドルウェアによってヘッドノードまたはディスプレイノードとして動作することが可能になる。ヘッドノードは各ディスプレイに画像を表示するための信号の送信やヘッドノード-ディスプレイノード間の同期処理などの役割を担う。ディスプレイノードはマルチディスプレイを構築するディスプレイに接続され画像の描画や表示に関する計算や処理を行う。ディスプレイノードとディスプレイは 1 対 1 で接続されるため、ディスプレイノードはマルチディスプレイを構築するディスプレイの数と同数にする必要がある。

以下、専用ミドルウェアを用いたマルチディスプレイシステムの詳細な動作について説明する。各ミドルウェアによって行われる連携表示処理には、大きく分けて 2 通りの手法がある。1 つはフレーム転送方式と呼ばれる手法である。フレーム転送方式はヘッドノードがディスプレイノードに対して表示映像のフレームを送信し、ヘッドノードとディスプレイノードの同期処理によってフレームをディスプレイ上に連携表示させる方式である。フレーム転送方式を採用するマルチディスプレイの例として、SAGE [5] や SAGE2 [6] などがあげられる。

フレーム転送方式は、以下の 5 種類の動作から成り立つ。

- フレーム圧縮
- フレーム送信

- フレーム展開
- 同期制御
- フレーム表示

まずヘッドノード上で表示映像からフレームが切り出され、フレームの圧縮処理が行われる。この時、フレームを各ディスプレイノードの表示領域に合わせて分割する手法と、フレーム全体をそのまま圧縮する手法の2通りが存在する。前者はSAGE [5]、後者はDisplayCluster [7]といったミドルウェアでそれぞれ採用されている。フレームの圧縮にはDXT (DirectX Texture Compression) [8] や JPEG (Joint Photographic Experts Group) [9] などが利用される。

続いて、ヘッドノードで圧縮されたフレームを各ディスプレイノードへ送信する。ディスプレイノードは、ヘッドノードから圧縮フレームを受信したあと展開処理を行い、続いて同期処理を開始する。同期処理では、まずフレームの展開を終え表示待機状態になったディスプレイノードからヘッドノードへ表示準備完了通知が送信される。ヘッドノードは、接続された全てのディスプレイノードから表示準備完了通知を受け取った後、各ディスプレイノード宛に表示命令を送信する。表示命令を受け取ったディスプレイノードは、該当フレームをディスプレイ上に表示する。この一連の動作を繰り返し行うことで、ディスプレイ上に連携して動画を表示することが可能になる。

もう1つは、描画命令転送方式と呼ばれる方式である。この方式では、ヘッドノードはディスプレイノードに対して画像フレームの送信を行わず、アプリケーションの出力する描画命令を同期処理を行いつつ送信することで動画の連携表示を行う。この方式を採用しているミドルウェアとしては、DMX (Distributed Multihead X) や Chromium [10] などがあげられる。

描画命令転送方式は、主に以下の4種類の動作から成り立つ。

- 描画命令のキャプチャリング
- 描画命令転送
- 同期制御
- フレーム描画

まず、ヘッドノードはアプリケーションから出力された描画命令のキャプチャリングを行う。続いて、ヘッドノードがキャプチャリングした描画命令をディスプレイノードへ送信する。描画命令を受信したディスプレイノードは、命令を受け取ったことを通知するメッセージをヘッドノードへ送信する。ヘッドノードは全てのディスプレイノードから描

画命令受信通知を受信し次第、フレーム描画命令を送信し、ディスプレイノードがこの描画命令を受信しディスプレイ上にフレームを描画する。この一連の動作を繰り返し行うことで、ディスプレイ上に連携して動画を表示する。

### シングルボードコンピュータ (SBC)

シングルボードコンピュータ (SBC) は、1枚の基板上に必要最低限の部品を取り付けることで構築された小型計算機である。SBC は他の汎用 PC と比較しても非常に低価格で手に入れることができ、消費電力も少ないため、IoT (Internet of Things) 向けの機器として一般に利用される [11, 12]。また、軽量プログラミング言語が利用できることから、教育用やプログラミング学習用としても使用されている。SBC には Raspberry Pi シリーズを代表として、様々な種類のものが存在する。表 2.1 に主な SBC の機種とその仕様を示す。また、図 2.3 に主な SBC の画像を示す。

表 2.1: SBC 機種の仕様および価格

機種	Raspberry Pi 4 Model B [13]	Orange Pi Zero Plus2 [14]	Banana Pi BPI-M64 [15]
CPU	ARM Cortex-72 (1.5 GHz × 4)	ARM Cortex-A53 (1.8 GHz × 4)	ARM Cortex-A53 (1.2 GHz × 4)
GPU	Broadcom VideoCore IV (500 MHz)	Mali T720MP2 (650 MHz)	Mali-400 MP2 (500 MHz)
メモリ	4.0 GB	4.0 GB	8.0 GB
通信帯域	1 Gbps	100 Mbps	1 Gbps
映像出力端子	HDMI × 2	HDMI × 1	HDMI × 1
OS	Linux	Linux	Linux
価格	約 8,000 円	約 14,000 円	約 9,000 円



Raspberry Pi Model 4 B [13]    Orange Pi Zero Plus2 [14]    Banana Pi BPI-M64 [15]

図 2.3: 主な SBC の画像

SBC が持つ CPU の特徴としては、主に低周波数のコアがデュアルコア、クアッドコアなどの形式で複数搭載されているものが多い。これは消費電力を抑えるため、また価格を低く抑えるためであり、こうした省電力や低価格という特徴を維持しつつ CPU 自体の処理能力を向上させる目的で、近年ではコア数の増加が進んでおり、マルチコア方式を採用した機種が多く販売されるようになってきている。搭載メモリや通信帯域に関しては、近年の SBC の性能向上により 4.0 GB 以上の比較的大きなメモリ容量を持つものも増えており、ギガビットイーサネットに対応した NIC を有するものも多く発売されるようになってきている。映像出力の機能としては、ほとんどの機種で HDMI 端子による出力が可能であり、Full HD 解像度での出力が標準的である。また、各機種の OS には軽量な Linux ベースの OS が広く採用されている。例えば Raspberry Pi に対応した OS である Raspbian [16] も Linux をベースとする OS の一種である。

## 2.2 先行研究で提案されたマルチディスプレイシステム

本節では、先行研究 [17] で提案された、SBC を用いて構築されたマルチディスプレイの特徴について述べる。まず、マルチディスプレイを構築する各ノードの動作について説明する。その後フレームの連携表示処理を可能にする仕組みについて概説する。

### 連携表示処理

先行研究で構築されたマルチディスプレイシステムは、1 台のヘッドノードと、SBC を用いて実装された複数台のディスプレイノードで構成されている。システムの概要を図 2.4 に示す。

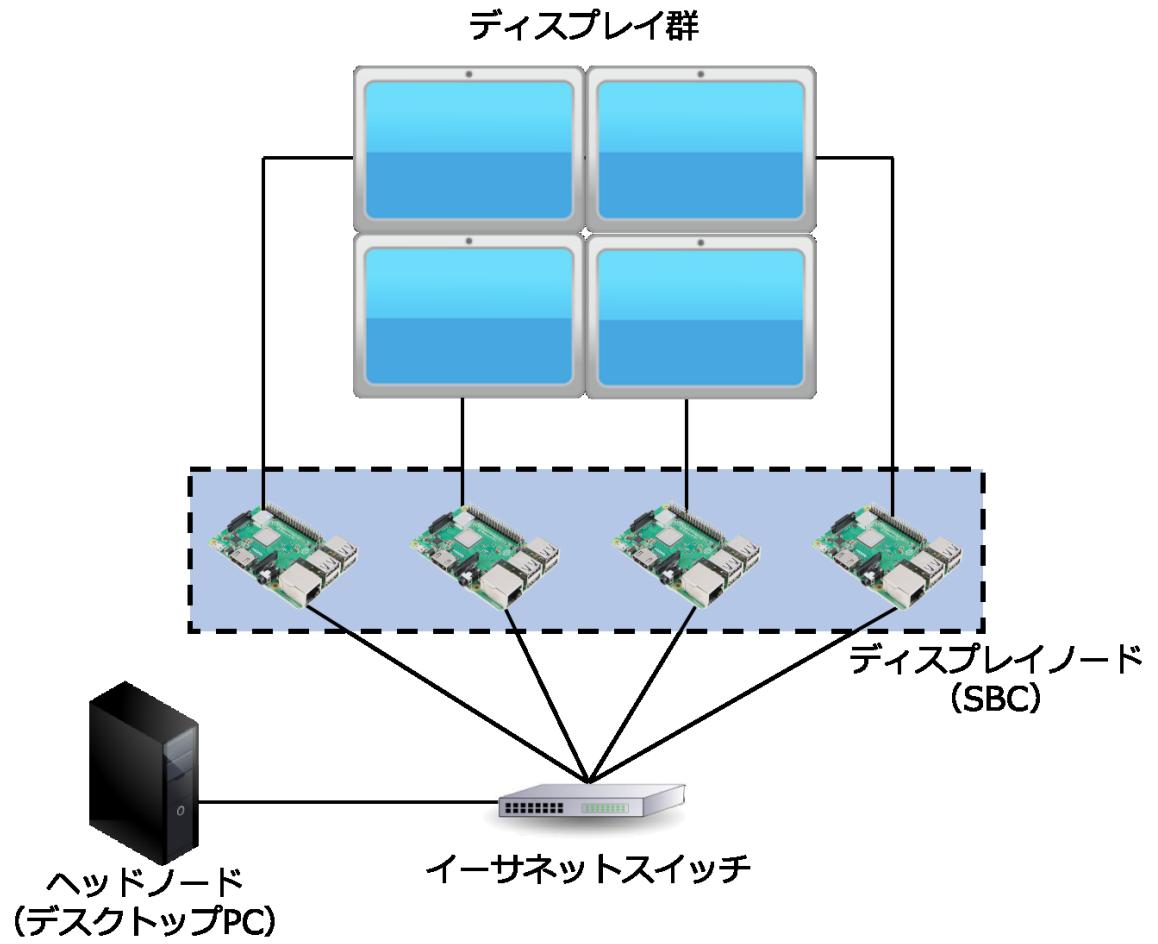


図 2.4: 先行研究システムの概要図

このマルチディスプレイシステムは、2.1.2節で述べた2種類の方式のうち、フレーム転送方式を採用している。そのため、SBCをディスプレイノードとして用いることとなり、従来のフレーム転送方式をそのまま適用するとSBC上で負荷の高い処理を行う必要がある。しかし、SBCは市販の汎用PCなどと比べると処理速度、描画性能などの面で劣っているため、可能な限りディスプレイノード上での処理負荷を小さく抑えなければならないという制約が存在する。そのため、SBCがディスプレイノードとして十分高速に動作するような設計を目的として、JPEG圧縮や連携表示処理のパイプライン化などが実装されている。

以下では、先行研究において提案されたマルチディスプレイシステムの詳細な動作について述べる。ヘッドノードでは、動画ファイルからフレームを切り出した後、接続ディスプレイノード数に応じてフレームの分割を行う。分割したフレームはそれぞれJPEG方式で圧縮され、各ディスプレイノードへと送信される。ヘッドノードから送信された圧縮フレームを受信したディスプレイノードでは、フレームを展開し、その展開が終了した後

表示バッファに格納してヘッドノードに表示準備完了メッセージを送信する。ヘッドノードは全てのディスプレイノードからの表示準備完了メッセージを受け取り次第表示命令を送信し、命令を受け取ったディスプレイノードはディスプレイ上にフレームを表示する。

また、ノード上の各処理はスレッド処理によって行われており、ヘッドノード上では圧縮スレッド、送信スレッド、同期制御スレッドの3種類、ディスプレイノード上では受信スレッド、展開スレッド、表示制御スレッドの3種類のスレッドが動作している。各スレッドの操作を図2.5に示す。圧縮スレッドは、表示映像の切り出しと分割、圧縮までを行う。送信スレッドは、圧縮されたフレームを各ディスプレイノードに送信する。同期制御スレッドは、表示準備完了メッセージの受信と表示命令の送信を行う。受信スレッドは、ヘッドノードから圧縮フレームを受信する。展開スレッドは、圧縮フレームの展開を行う。表示スレッドは、ヘッドノードからの表示命令を受け取りフレームをディスプレイ上に表示する。

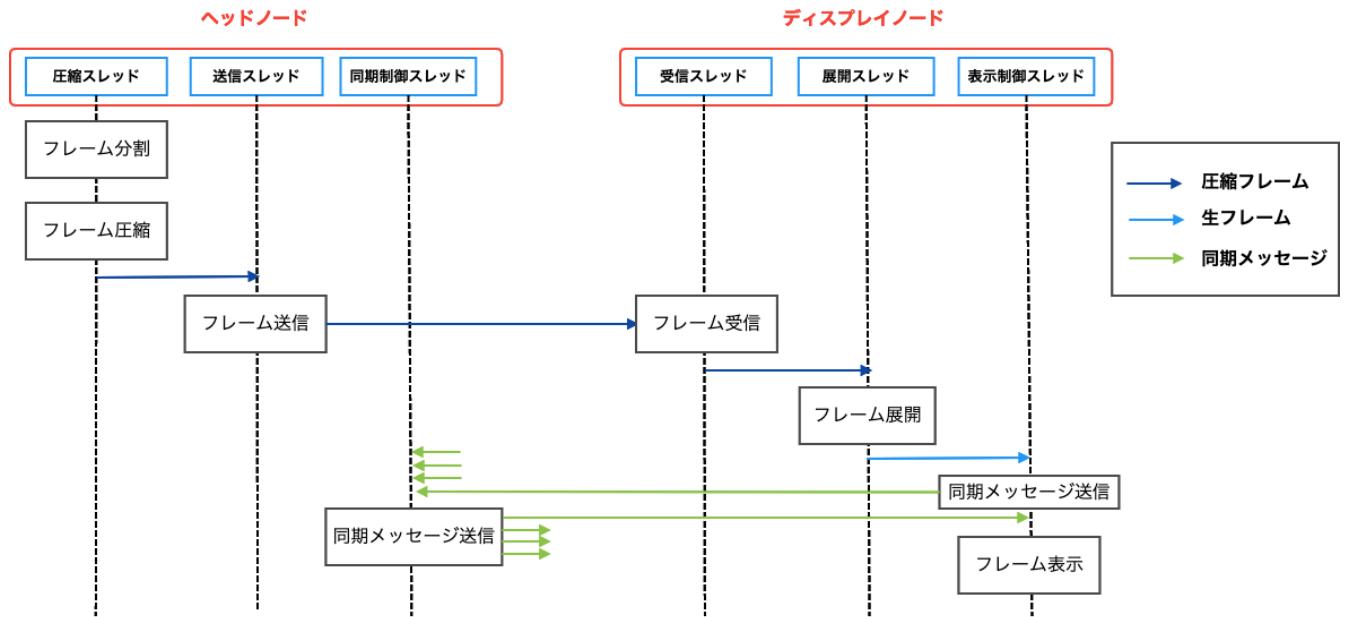


図 2.5: 各スレッドの動作

## フィードバック処理

JPEG圧縮処理では、まず入力画像のピクセルごとの画素値をRGB形式からYCbCr形式に変換する。YCbCrは色彩表現法の一種であり、色を人間の目が認識しやすい輝度成分と認識しづらい色差成分とに分けて表現する[18]。YCbCr形式に変換された色成分はデータ量を削減するために間引かることになるが、この時YCbCrサンプル比という

間引き方を決定するパラメータを変更することで、JPEG圧縮に要する時間を変更することができる [19]. また、量子化を行う際の品質係数と呼ばれるパラメータを変更することによっても、JPEG圧縮に要する時間を短縮することができる。先行研究のマルチディスプレイシステムでは、ディスプレイノードで動画再生時のフレームレートを計算し、目標フレームレートとの差に応じてこれらのパラメータを変更することで、フレームレートを目標値に近づけるフィードバック処理を実装している。

## 問題点と技術的課題

本節では、前節で述べた先行研究におけるマルチディスプレイシステムの問題点と、それを解決するための技術的課題について述べる。まず、先行研究のマルチディスプレイシステムにおける具体的な問題について説明する。その後、問題点解決に向けた手法の考察と利点、欠点について述べ、比較を行う。

### 先行研究で提案されたマルチディスプレイシステムの問題点

先行研究で構築されたマルチディスプレイには、高解像度な動画や高フレームレートな動画を表示しようとするとパフォーマンスが低下するという問題点がある。この問題の原因となっているのが、ヘッドノードでのフレーム処理である。前述したように、ヘッドノード内では動画フレームに対して分割、圧縮、送信の3つの処理が行われている。高解像度な動画では1フレームあたりのデータ量が大きくなるためフレームの圧縮に時間がかかり、パフォーマンスが低下する。また、高フレームレートな動画ではフレーム圧縮に要する時間が動画における1フレームの表示時間を上回り、動画本来のフレームレートを維持したまま動画を再生することが困難になる。

また、大規模なマルチディスプレイを構築した際のパフォーマンス低下も問題点として挙げられる。先行研究で構築されたMDの中では圧縮スレッド、送信スレッド、同期制御スレッドがそれぞれ1つのスレッドとして動作している。マルチディスプレイを構成するディスプレイの数が増加した場合にはそれに伴ってフレーム圧縮処理の回数も増加し、圧縮スレッドでのフレーム処理負荷も同様に増加することとなる。しかし、ヘッドノード内で動作する圧縮スレッドは1つであるため、処理負荷の増加に伴って処理に要する時間も増加することとなり、動画再生時のフレームレート低下につながる。実際、4面構成の場合では30fpsの動画をMD上に20~22fpsで再生することが可能だが、ディスプレイを9面構成にしたMDでは8~10fpsに低下することが確認できる(図2.6)。

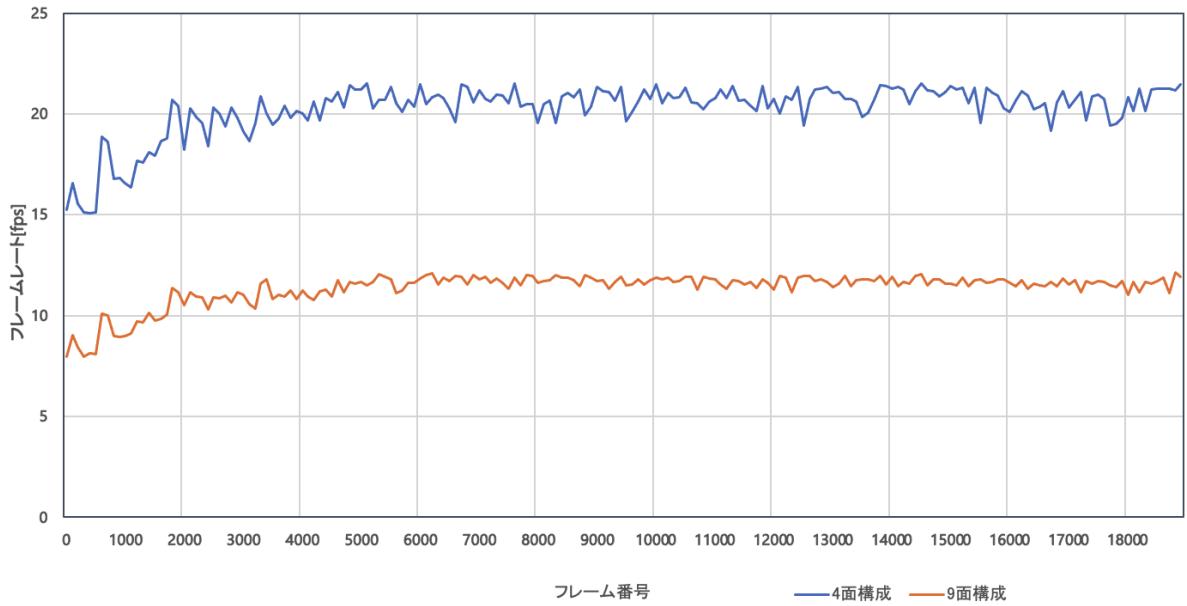


図 2.6: 画面構成変更時のフレームレート比較

### 2.3 ヘッドノードの処理負荷軽減を行った関連研究

ヘッドノードでのフレーム処理負荷が増加しフレームレートが低下するという問題に対して有効な対策として、ヘッドノードでのフレーム処理を並列化することにより処理負荷を軽減し、フレーム処理に要する時間の短縮を試みる手法が考えられる。この手法を実装した関連研究として、自身の卒業研究がある。

#### 実装の概要

関連研究では、ヘッドノードでの処理を分散並列化する目的で、処理のノード分散を行っている。ノード分散では、新たにノードとしてCPUを追加実装することによって物理的にCPUを増加させ、高負荷なフレーム処理を複数のCPUで分担して実行する。また、ヘッドノードの性能に依存することが少なくなるため、ヘッドノードとしてCPUコア数が少ないような比較的低価格なPCを用いた場合にも大きく性能を落とさずに動作することが期待できる。

関連研究では、ヘッドノードの処理を複数のノードに分散するために、ヘッドノードとディスプレイノードとの間に新たにSBCを用いて中継ノードを実装した。また、連携表示処理の並列化を行うため、中継ノードでの処理を受信スレッド、画像処理スレッド、送信スレッドという3種類のスレッドで分担して行うようマルチスレッド化している。各スレッドの動作を図2.7に示す。

## 中継ノードの動作

受信スレッドはヘッドノードから送信される一次分割済み非圧縮画像フレームを受信し、受信バッファへの格納を行う。分割・圧縮スレッドは受信バッファに格納された非圧縮フレームを取り出し、接続されているディスプレイノードの数に応じてフレームの二次分割処理を行う。その後、それぞれの分割済フレームに対して JPEG 圧縮処理を行い、送信バッファへと格納する。送信スレッドは、送信バッファに格納された JPEG 圧縮済フレームを、各接続ディスプレイノードへと送信する処理を行う。以上の 3 種類のスレッドによって、中継ノードはヘッドノードとディスプレイノードとの間で連携表示処理を実現する。

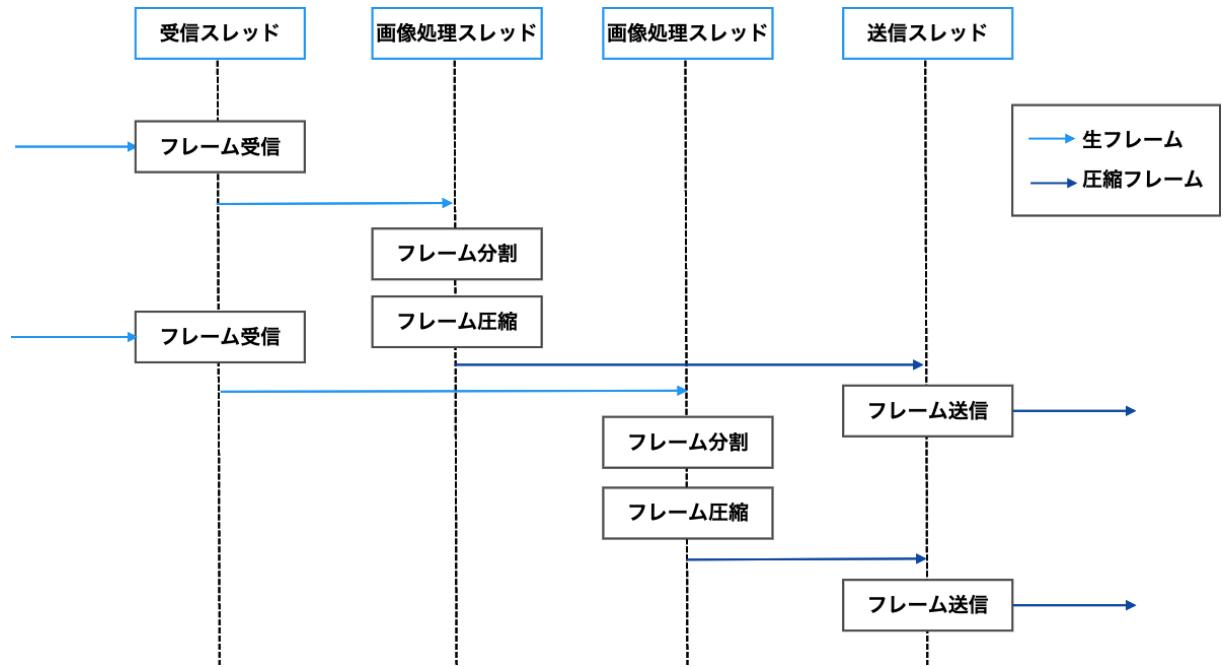


図 2.7: 各スレッドの動作

## ヘッドノードの動作

中継ノードを実装したマルチディスプレイシステムでは、ヘッドノードは接続中継ノード数に応じて画像フレームの分割を行う。分割された画像フレームは、送信スレッドへ渡され、各フレームに対してフレーム番号が付与される。そして、フレーム番号の付与が終了したフレームは圧縮処理を行わずに中継ノードへと送信され、中継ノードは非圧縮フレームを受信して受信バッファに格納する。その後、画像処理スレッドが非圧縮フレームを受信バッファから取り出し、接続ディスプレイノード数に応じて分割し、パラメータに従い JPEG 形式に圧縮してディスプレイノードへ送信する。

## ディスプレイノードの動作

ディスプレイノードは中継ノードから送信された圧縮フレームを受信し、受信バッファに格納する。その後、展開スレッドが圧縮フレームを展開する。展開処理が終了したフレームは表示バッファに格納され、表示制御スレッドがこれを認識してヘッドノードへ表示準備が完了した旨の同期メッセージを送信する。そして、ヘッドノードからの表示命令を受け、表示制御スレッドが表示フレームの切り替えを行うことにより順次フレームがディスプレイ上に表示される。この一連の動作を繰り返すことにより、マルチディスプレイシステムとしての連携表示処理が可能となる。

図3.2に提案手法を用いて構成した4面構成のマルチディスプレイシステム全体の論理構成図と動作フロー図を示す。

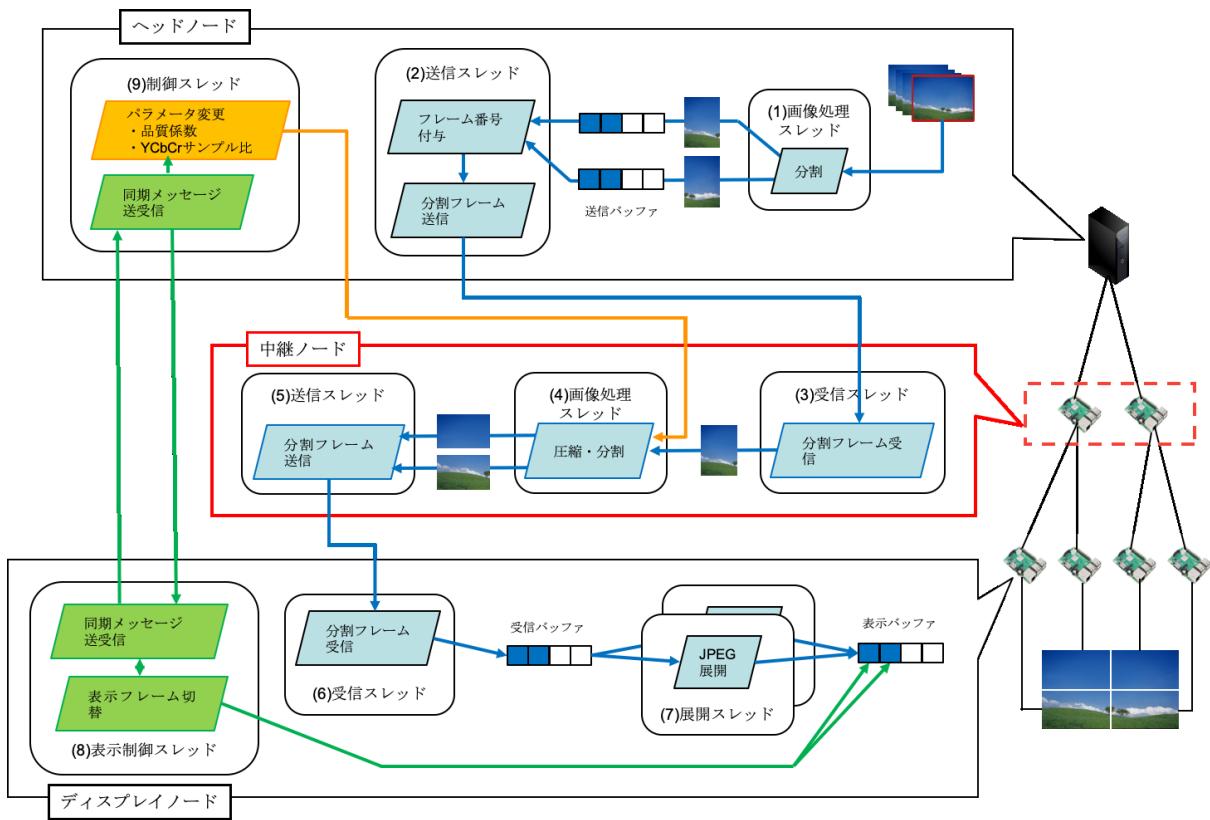


図 2.8: 関連研究で提案された手法を用いた MD の動作フロー

## 関連研究の問題点と将来課題

ヘッドノードとディスプレイノードの間に中継ノードを実装してフレーム処理の分散並列化を図った関連研究では、ヘッドノードの高負荷状態を解消する事ができた。しかし、フレームレートなどシステムのパフォーマンスの面では、元々のマルチディスプレイシステムと比較して性能が低下した。原因として考えられるのは、中継ノードを新たに実装したことで生じたフレームの送受信処理である。関連研究で提案されたマルチディスプレイ

システムでは中継ノード内で画像フレームの圧縮処理を行うため、画像フレームはまずヘッドノードから中継ノードへと送信され、中継ノードで圧縮処理がなされた後ディスプレイノードへと送信される。この2度のフレーム送受信処理によって生じるオーバーヘッドが影響し、システムのパフォーマンスが低下したと考えられる。

## 2.4 本研究での技術的課題

関連研究の結果から、ヘッドノードの処理を外部のノードへ分散させる手法は画像フレームデータの送受信のオーバーヘッドが生じるためパフォーマンスの向上への貢献度が低い事がわかった。フレームデータ送受信のオーバーヘッドを生じさせずにフレーム処理の並列化を行うためには、ヘッドノード内のCPUリソースを活用する事が必要である。

### 3 提案

提案手法について簡単に書く OS 仮想化基盤を用いた SBC マルチディスプレイシステムのフレーム処理並列化

#### 3.1 OS 仮想化基盤とコンテナ技術

OS 仮想化基盤には、代表的なものとして Docker [20] が挙げられる。

#### 3.2 MD のコンテナ化

以下、コンテナ技術を用いた SBC マルチディスプレイシステムの実装について書く

#### 3.3 コンテナを通したディスプレイの制御

コンテナ内からディスプレイを制御する方法について書くフレームバッファを利用したディスプレイ表示のメカニズムについて図を使って解説できると良い

#### 3.4 ヘッドノードでのフレーム処理の改良

ヘッドノード内のコンテナを用いたフレーム処理の並列化について書くメインコンテナ、圧縮コンテナの実装や動作などについて図やフローチャートを用いて解説する

フレーム処理のコンテナ並列化

コンテナ間でのフレーム受け渡し処理

System V IPC [21]



図 3.1: 共有メモリの使用例

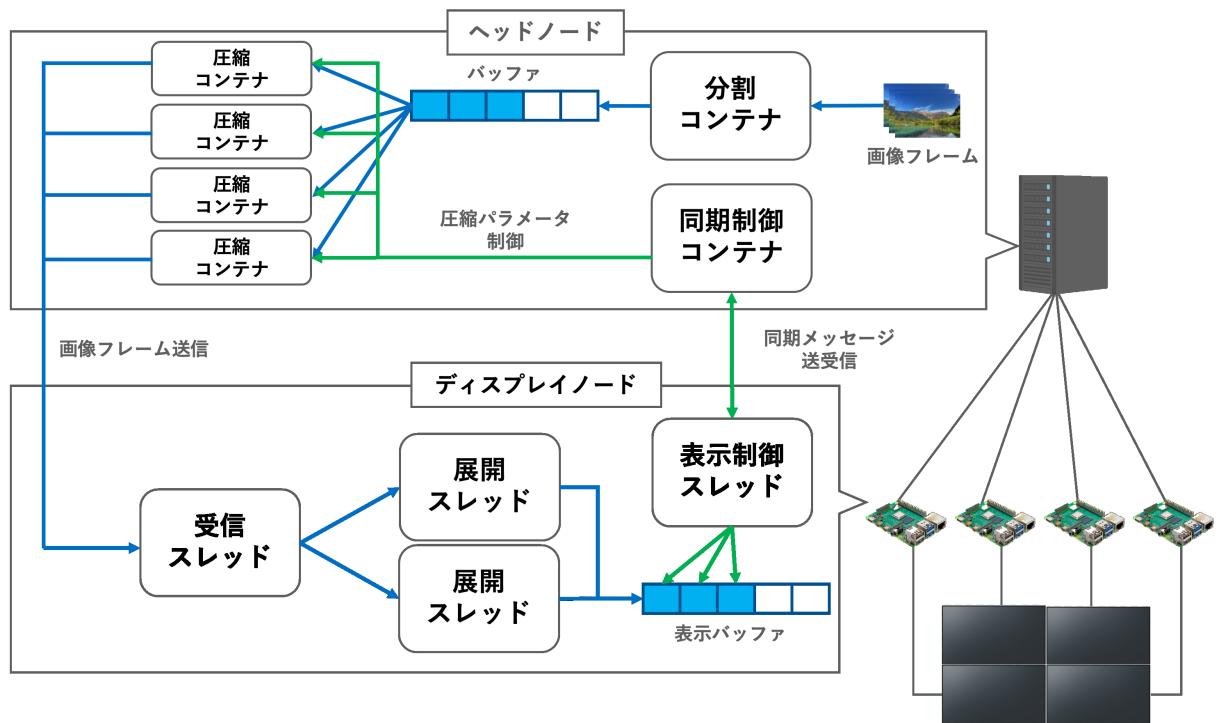


図 3.2: 提案手法を用いた MD の動作フロー

## 4 評価

行った評価の内容について簡単に説明する

### 4.1 実験環境

実験環境について書く使用機器，使用ネットワーク帯域，使用動画ファイル，OS，モジュールなど

表 4.1: ヘッドノード用デスクトップPCの仕様

要素	仕様
CPU	i7-5960X (3.0 GHz × 8)
メモリ	64.0 GB
通信帯域	1 Gbps
OS	CentOS 7.3

評価に用いる映像としては，クリエイティブ・コモンズ・ライセンスのもとで利用できる3DアニメーションであるBig Buck Bunny [22]を使用した。表4.4に，Big Buck Bunnyのプロパティを示す。さらに，図4.3にBig Buck Bunny再生時のスクリーンショットを示す。

表 4.2: Big Buck Bunny のプロパティ

項目	内容
動画形式	MP4
コーデック	H.264 [23]
長さ	10 分 34 秒
総フレーム数	19020 枚
フレームレート	30 fps
解像度	4K (3840 × 2160)



図 4.1: Big Buck Bunny のスクリーンショット

ヘッドノードで行われる処理の実装には，OpenCV [24]，libjpeg-turbo [25]，Boost.Asio [26] という 3 種類のライブラリを利用した。OpenCV は画像処理ライブラリ，libjpeg-turbo は JPEG の圧縮と展開を行うライブラリ，Boost.Asio はソケット通信を行うライブラリである。また，ディスプレイノードの処理の実装には libjpeg-turbo，Boost.Asio，fbdev を利用した。実装には C++ 言語を使用し，GCC コンパイラ (GNU C Compiler) [27] でコンパイルを行った。

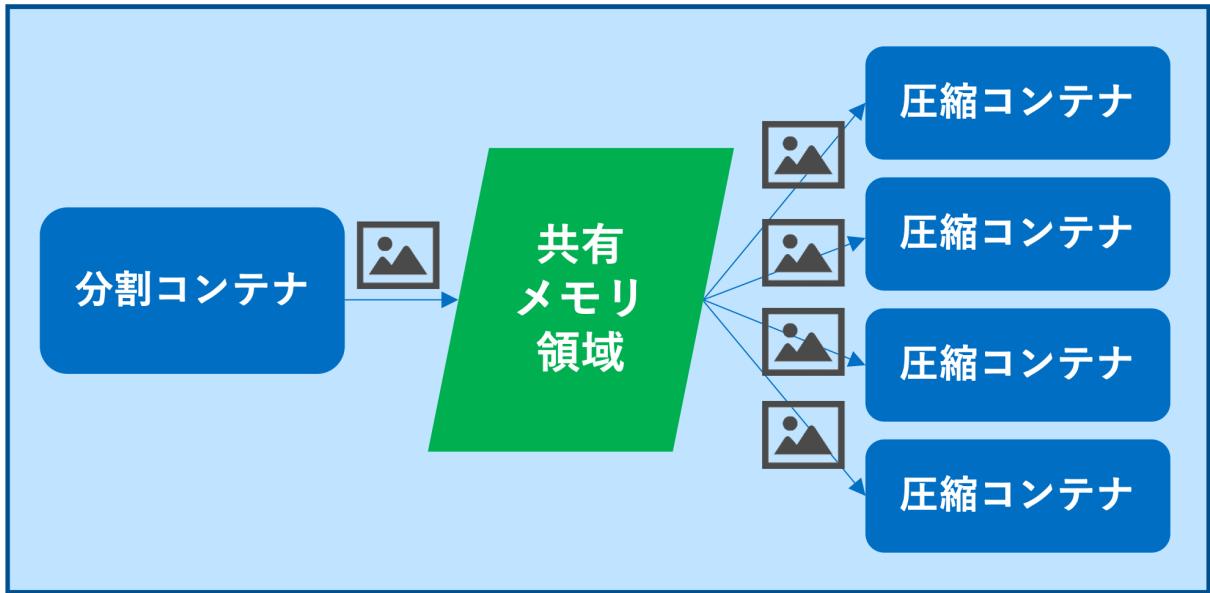


図 4.2: 実験環境のコンテナ構成

## 4.2 フレーム処理時間の評価

### 4.2.1 実験方法

先行研究で提案された手法と提案手法において、4面構成のMDを想定した環境で4K解像度の画像を表示し、動画のフレーム開始から1000フレーム目の処理が終了するまでの1フレームあたりに要するフレーム処理時間を計測した。図4.6に先行研究で提案されたシステムと提案手法でのフレーム処理時間を示す。

### 4.2.2 実験結果

既存手法ではヘッドノードでのフレーム処理時間の平均が52.35msであったのに対し、提案手法では21.72msとなっており、フレーム処理時間が既存手法と比較して40%程度まで短縮された事が確認できた。

また、本提案で新たに生じた処理である共有メモリを用いたプロセス間での画像フレームデータ受け渡しに要する時間も平均で1ms未満となっており、オーバーヘッドは非常に小さくなっていることも確認できる。

以上の結果より、フレーム処理プロセスの並列化を行った提案手法によって、システムのボトルネックが解消された。

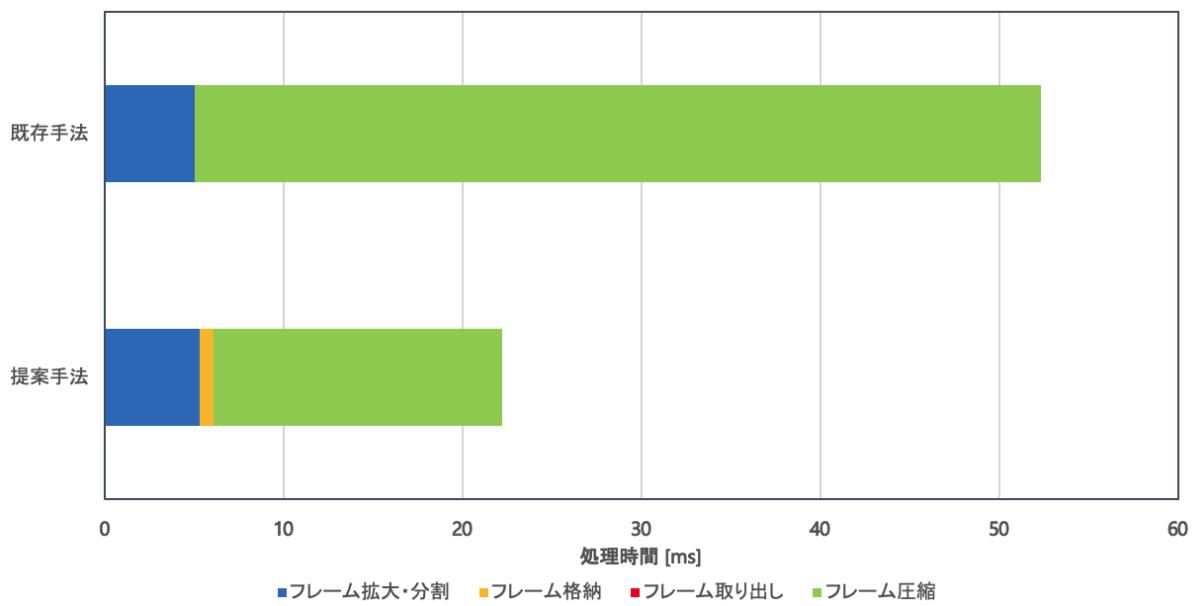


図 4.3: フレーム処理時間の比較

### 4.3 フレーム処理速度の評価

#### 4.3.1 実験方法

#### 4.3.2 実験結果

#### 4.3.3 結果に対する考察

## 5 結論

## 謝辞

本研究を行うにあたり、懇切なる御指導、御鞭撻を賜りました大阪大学サイバーメディアセンター 下條真司教授に心より感謝の意を表します。

本研究の全過程を通じ、研究の方向性検討や論文の執筆等、熱心な御指導、御鞭撻を賜りました大阪大学サイバーメディアセンター応用情報システム研究部門 木戸善之講師に心より感謝申し上げます。

本研究の進行から論文執筆まで多岐にわたり、手厚い御指導、御鞭撻を賜りました大阪大学サイバーメディアセンター応用情報システム研究部門 伊達進准教授、小島一秀講師に心より感謝の意を表します。

本研究において、種々のご意見、ご指導を賜りました大阪大学情報推進本部 大平健司准教授に心より感謝いたします。

本研究において、厳しくも優しい御指導を賜りました大阪大学データビリティフロンティア研究機構 春本要教授に心より感謝の意を表します。

最後に、日々御支援と心からの激励をかけて頂いた下條研究室の皆様ならびに事務補佐員 片岡小百合氏に心より感謝いたします。

## 参考文献

- [1] Luciano Soares, Bruno Raffin, and Joaquim Jorge. PC clusters for virtual reality. *IJVR*, Vol. 7, pp. 67–80, Jul. 2008.
- [2] H. Chung, C. Andrews, and C. North. A survey of software frameworks for cluster-based large high-resolution displays. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 20, No. 8, pp. 1158–1177, Aug. 2014.
- [3] 中村伊知哉. デジタルサイネージの動向. 情報管理, Vol. 55, No. 12, pp. 891–898, Mar. 2013.
- [4] 長野英生. Displayport. 映像情報メディア学会誌, Vol. 66, No. 11, pp. 942–945, May 2012.
- [5] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, G. Goldmana, J. Leigh, and A. Johnson. SAGE : the scalable adaptive graphics environment. *Proceedings of the 4th Workshop on Advanced Collaborative Environments (WACE2004)*, Sep. 2004.
- [6] Luc Renambot, Thomas Marrinan, Jillian Aurisano, Arthur Nishimoto, Victor Mattevitsi, Krishna Bharadwaj, Lance Long, Andy Johnson, Maxine Brown, and Jason Leigh. SAGE2: A collaboration portal for scalable resolution displays. *Future Generation Computer Systems*, Vol. 54, pp. 296–305, Jan. 2016.
- [7] G. P. Johnson, G. D. Abram, B. Westing, P. Navr’til, and K. Gaither. Displaycluster: an interactive visualization environment for tiled displays. In *Proceedings of the 16th International Conference on Cluster Computing*, pp. 239–247, Sep. 2012.
- [8] Petr Holub, Martin Srom, Martin Pulec, Jiri Matela, and Martin Jirman. GPU-accelerated DXT and JPEG compression schemes for low-latency network transmissions of HD, 2K, and 4K video. *Future Generation Computer Systems*, Vol. 29, pp. 1991–2006, Oct. 2013.
- [9] G. K. Wallace. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics*, Vol. 38, No. 1, pp. 18–34, Feb. 1992.
- [10] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH*, Vol. 21, pp. 693–702, Jul. 2002.

- [11] 小林敬. IoT を指向したシングルボードコンピュータの食品成分分析およびセンシングへの応用. 日本食品工学会誌, Vol. 20, No. 3, pp. 107–113, Oct. 2019.
- [12] M. S. D. Gupta, V. Patchava, and V. Menezes. Healthcare based on IoT using Raspberry Pi. In *Proceedings of the International Conference on Green Computing and Internet of Things (ICGCIoT2015)*, pp. 796–799, Oct. 2015.
- [13] Raspberry Pi 4 Model B. <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>, Feb. 2020.
- [14] Orange Pi Zero Plus2. <http://www.orangepi.org/OrangePiZeroPlus2/>, Feb. 2020.
- [15] Banana Pi オープンソースプロジェクト. <http://www.banana-pi.org/m64.html>, Feb. 2020.
- [16] Raspbian. <https://www.raspbian.org/>, Feb. 2020.
- [17] 石田和也. 小型低性能算機向けマルチディスプレイ構築用ミドルウェア, Feb. 2019.
- [18] Noor Ibraheem, Mokhtar Hasan, Rafiqul Zaman Khan, and Pramod Mishra. Understanding color models: A review. *ARPN Journal of Science and Technology*, Vol. 2, pp. 265–275, Jan. 2012.
- [19] P. T. Chiou, Y. Sun, and G. S. Young. A complexity analysis of the JPEG image compression algorithm. In *Proceedings of the 9th Computer Science and Electronic Engineering (CEEC2017)*, pp. 65–70, Sep. 2017.
- [20] Empowering app development for developers — docker, Feb. 2022.
- [21] Michael Kerrisk. *The Linux programming interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010.
- [22] Big Buck Bunny. <https://peach.blender.org/>, Feb. 2020.
- [23] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 560–576, Jul. 2003.
- [24] OpenCV. <https://opencv.org/>, Feb. 2020.
- [25] libjpeg-turbo. <https://libjpeg-turbo.org/>, Feb. 2020.
- [26] boost C++ libraries. <https://www.boost.org/>, Feb. 2020.
- [27] GCC,GNU コンパイラコレクション. <https://gcc.gnu.org/>, Feb. 2020.