+++++++++++++++++++++++++++++++++++++

src/switch

+++++++++++++++++++++++++++++++++++++

\chapter{SWITCH 方式}

\label{sec:switch}

従来のIQでは，空きエントリがあればディスパッチすることができたが，これまで提案した手法では，ディスパッチできるエントリは選択されたセグメントに限定されている．このためIQ の容量効率が低下するという問題がある．本章では，この問題に対応する手法である SWITCH 方式に関して説明する．\refsec{occupency\_reduction}で IQ の容量効率の低下に関して説明した後，\refsec{switch\_scheme}で SWITCH 方式に関して説明する．

\begin{figure}[tb]

\centering

\includegraphics[keepaspectratio, scale=.8]{stall\_segmentedIQ}

\caption{容量効率が低下する例}

\label{fig:stall\_segmentedIQ}

\end{figure}

\section{容量効率の低下}

\label{sec:occupency\_reduction}

提案手法には IQ の容量効率が低下するという問題がある．この問題は，IQ の容量効率が重要なプログラムにおいて，性能低下を引き起こす．本節では，容量効率が低下する原因について説明した後，容量効率の低下により性能低下を引き起こすプログラムの特徴に関して説明する．

\subsection{提案手法による容量効率低下の原因}

IQ の容量効率の低下に関して，\fig{stall\_segmentedIQ}を用いて説明する．図において，灰色のエントリは命令を保持していることを示している．

図の状態の IQ に，新たに命令 $p2 = p5 + p6$ をディスパッチする場合を考える．この命令のソース・オペランドは両方レディでないとする．この場合，第 1 ソース・タグの下位ビットから第 1 セグメントにディスパッチされることが決定する．しかし，第 1 セグメントに空きエントリはないため，空きが出るまでディスパッチをストールさせ，待ち合わせを行う必要がある．

このように，命令がディスパッチされるセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチできないため，提案手法ではセグメント化されていない IQ と比較して容量効率が低下する．

\subsection{容量効率の低下による性能低下}

プログラムには，性能が IQ の容量に敏感なものとそうでないものとがある~\cite{Ando2019, Kora2013, Sembrant2015}.次の 2 つの特徴のうちいずれかに当てはまるプログラムでは，性能が IQ の容量に敏感なため，与えられた IQ の容量においては，その利用効率が重要である．このため，提案手法による容量効率の低下によって性能が低下する．

\begin{itemize}

\item \textbf{命令レベル並列性（ILP：Instruction Level Parallelism）}が高いプログラム

\item \textbf{メモリ・レベル並列性（MLP：Memory Level Parallelism）}が高いプログラム

\end{itemize}

ILP が高いプログラムでは， IQ に命令をできるだけ多く供給し，より多くの命令を並列に発行できるようにすることで高い性能が得られる．IQ の容量効率が低下すると，並列に発行できる命令数が減少するため，性能が低下する．

MLP が高いプログラムでは，できるだけ多くのキャッシュ・ミスを並列に実行することにより，メモリ・アクセスのレイテンシが実行時間に与える影響を縮小できる．IQ の容量効率が低下すると，並列に処理できるメモリ・アクセスが減少するため，性能が低下する．

これらのことから，ILP もしくは MLP が高い場合には，提案手法による容量効率の低下を最小限に抑える工夫が必要となる．

%5 容量効率の低下への対策：SWITCH 方式

\section{容量効率低下への対策：SWITCH 方式}

\label{sec:switch\_scheme}

IQ の容量効率低下による性能低下を抑制する方式として，\textbf{SWITCH} と呼ぶ方式を提案する．SWITCH 方式では，次のようにして性能低下を抑制する．

\begin{itemize}

\item セグメント選択回路の選択アルゴリズムとして，容量効率は低下するが，タグ比較回数を多く削減できるような選択を行う \textbf{AGGRESSIVE モード} と，タグ比較回数の削減率は低下するが，容量効率が大きく低下しないような選択を行う \textbf{CONSERVATIVE モード} の 2 つを用意する．

\item 実行プログラムの ILP 及び MLP を一定のインターバルで監視し，ILP もしくは MLP が高いと判断されたなら次のインターバルでは CONSERVATIVE モードでディスパッチし，そうでないなら AGGRESSIVE モードでディスパッチを行う．

\end{itemize}

本節では，まず 2 つのセグメント選択のアルゴリズムに関して説明を行う．その後，ILP 及び MLP の評価方法と，切り替えアルゴリズムに関して説明する．

\subsection{2 つのセグメント選択アルゴリズム}

\label{sec:two\_mode}

SWITCH 方式では，タグ比較回数の削減重視の AGGRESSIVE モードと，容量効率重視の CONSERVATIVE モードの 2 つを適切に切り替えて使用する．各モードには，タグ比較回数の削減と容量効率に関して，\tab{switch\_trade\_off}に示すトレード・オフの関係がある．それぞれのセグメントの選択方法に関して説明する．

なお， SWITCH 方式はサブ・セグメントと併用が可能である．まず，サブ・セグメントを使用しない場合の AGGRESSIVE 及び CONSERVATIVE のアルゴリズムに関した説明した後，サブ・セグメントと併用する場合のアルゴリズムへと拡張して説明する．

\begin{table}[tb]

\caption{2 つのセグメント選択モードのトレード・オフ}

\footnotesize

\center

\begin{tabular}{l|c|c} \hline \hline

モード & タグ比較回数の削減 & 容量効率 \\ \hline

AGGRESSIVE & ○ & × \\

CONSERVATIVE & × & ○ \\ \hline

\end{tabular}

\label{tab:switch\_trade\_off}

\end{table}

\subsection{AGGRESSIVE モード}

AGGRESSIVE モードは，\tab{agg\_algorithm}で示した選択アルゴリズムを使用してディスパッチするエントリを決定する．このモードでは，選択されたセグメントに空きがない場合，他のセグメントに空きがあってもディスパッチは行わないため，容量効率が低下する．しかし，セグメント化の利益を最大限利用し，タグ比較回数を大幅に削減できる．

\subsection{CONSERVATIVE モード}

AGGRESSIVE モードでは，命令のソース・オペランドが両方レディであり，セグメント・インディペンデントとしてディスパッチできる場合以外では，ソース・タグによって選択されるセグメントに空きがない場合にディスパッチをストールさせる．これに対し，CONSERVATIVE モードでは，以下で説明する工夫を行うことによって，このディスパッチのストールを回避し，容量効率の低下を抑制する．

\begin{itemize}

\item \textbf{両ソース・オペランドともレディでない場合}：\\AGGRESSIVE モードでは第 1 ソース・タグの下位ビットによってセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがない場合には，\textbf{スワップ}~\footnote{\ref{sec:swap}節では，スワップの定義を「第 1 ソース・オペランドのみレディの場合に，第 1 ソース・タグと第 2 ソース・タグを書き込むフィールドを交換する」としていたが，本節以降ではこの定義を拡大し，単に「第 1 ソース・タグと第 2 ソース・タグを書き込むフィールドを交換する」という意味で用いる．}\textbf{して}ディスパッチを試みる．スワップするため，第 2 ソース・タグにより選択されるセグメントに空きがあればディスパッチが可能となる．

\item \textbf{第 1 ソース・オペランドのみレディである場合}：\\AGGRESSIVE モードでは，スワップを行い，第 2 ソース・タグでセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 2 ソース・タグによって選択されたセグメントに空きがない場合には，\textbf{スワップをやめて}ディスパッチする．スワップをやめるため，第 1 ソース・タグによってセグメントが選択されようとするが，第 1 ソース・オペランドは既にレディであるため，どのセグメントにディスパッチしても良い．従って，メイン・セグメント・インディペンデントとしてディスパッチが可能となる．

\item \textbf{第 2 ソース・オペランドのみレディである場合}：\\AGGRESSIVE モードでは第 1 ソース・タグでセグメントを選択する．選択されたセグメントに空きがない場合，ディスパッチを行わない．これに対して CONSERVATIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがない場合には，\textbf{スワップして}ディスパッチする．スワップするため，第 2 ソース・タグによってセグメントが選択されようとするが，第 2 ソース・オペランドは既にレディであるため，どのセグメントにディスパッチしても良い．従って，メイン・セグメント・インディペンデントとしてディスパッチが可能となる．

\end{itemize}

上述の工夫によって，CONSERVATIVE モードでは，どちらかのソース・オペランドがレディである場合は，必ずディスパッチが可能となる．また，両ソース・オペランドともレディでない場合でも，第 1 ソース・タグにより選択されるセグメントと第 2 ソース・タグにより選択されるセグメントのうち，いずれかのセグメントに空きがあればディスパッチが可能となる．従って，ストールする確率は大きく減少する．

\tab{cons\_algorithm}に，CONSERVATIVE モードにおけるセグメントの選択アルゴリズムをまとめる．

\begin{table}[htb]

\caption{CONSERVATIVE モードのアルゴリズム}

\footnotesize

\center

\begin{tabular}{|c|p{13.5cm}|} \hline \hline

ソース・タグの状態 & アルゴリズム \\ \hline

（NR，NR） & 第 1 ソース・タグでセグメントを選択．選択したセグメントに空きがない場合， スワップして第 2 ソース・タグをもとにセグメントを決定．なおも空きがない場合はストール． \\ \hline

（R，NR） & スワップを行い，第 2 ソース・タグでセグメントを選択．選択したセグメントに空きがない場合，スワップをやめてメイン・セグメント・インディペンデントとしてセグメントを選択する．\\ \hline

（NR，R） & 第 1 ソース・タグでセグメントを選択．選択したセグメントに空きがない場合，スワップを行いメイン・セグメント・インディペンデントとしてセグメントを選択．\\ \hline

（R，R） & セグメント・インディペンデントとしてセグメントを選択． \\ \hline

\end{tabular}

\label{tab:cons\_algorithm}

\end{table}

\subsubsection{CONSERVATIVE モードにおけるタグ比較回数の削減}

CONSERVATIVE モードでは，AGGRESSIVE モードと比較してタグ比較回数の削減率が 低下する可能性がある．この理由について説明する．例として，第 2 ソース・オペランドのみレディである命令をディスパッチする場合について説明する．

CONSERVATIVE モードでは，まず第 1 ソース・タグでセグメントを選択する．選択されたセグメントに空きがあれば，そのセグメントにディスパッチする．この場合，レディでない第 1 ソース・タグが，セグメント化によってタグ比較回数が削減される第 1 ソース・タグのフィールドに書き込まれるため，AGGRESSIVE モードと同様にタグ比較回数が削減される．

第 1 ソース・タグによって選択されたセグメントに空きがなければ，CONSERVATIVE モードではスワップしてセグメント・インディペンデントとしてディスパッチする．この場合，タグ比較回数の削減は行うことができない．これは，既にレディである第 2 ソース・オペランドのタグが，セグメント化によってタグ比較回数を削減できる第 1 ソース・タグのフィールドに書き込まれ，一方で，まだレディでなくタグ比較が行われる第 2 ソース・オペランドのタグが，セグメント化によってタグ比較回数が削減されない第 2 ソース・タグのフィールドに書き込まれるためである．

AGGRESSIVE モードでは，第 1 ソース・タグによって選択されたセグメントに空きがなければ，ストールして空きが出るまで待ち合わせる．このストールにより，容量効率は低下するが，空きが出たあとディスパッチするため，タグ比較回数は削減される．これに対して CONSERVATIVE モードでは，タグ比較回数の削減は行えなくなるが，スワップしてディスパッチすることによってストールを回避し，容量効率の低下を防ぐ．

従って，CONSERVATIVE モードは，タグ比較回数の削減をある程度犠牲にして，IQ の容量効率の低下を抑制するアルゴリズムであるといえる．

\subsection{サブ・セグメントとの併用}

SWITCH 方式とサブ・セグメントを併用する際の，AGGRESSIVE と CONSERVATIVE のディスパッチ・アルゴリズムに関して説明する．

AGGRESSIVE モードに関しては，\tab{agg\_algorithm\_subseg}で示したアルゴリズムがそのまま サブ・セグメントを併用する場合の AGGRESSIVE モードでのアルゴリズムとなる．

CONSERVATIVE モードのアルゴリズムを，\tab{cons\_algorithm\_subseg}に示す．

\begin{table}[htb]

\caption{CONSERVATIVE モードのアルゴリズム（サブ・セグメントと併用）}

\footnotesize

\center

\begin{tabular}{|c|p{13.5cm}|} \hline \hline

ソース・タグの状態 & アルゴリズム \\ \hline

（NR，NR） & 第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択．選択したセグメントに空きがない場合， スワップして第 2 ソース・タグでメイン・セグメントを，第 1 ソース・タグでサブ・セグメントを決定．なおも空きがない場合はストール． \\ \hline

（R，NR） & スワップを行い，第 2 ソース・タグでメイン・セグメントを選択し，サブ・セグメントはS-seg インディペンデントとして選択．選択したセグメントに空きがない場合，スワップをやめ，第 2 ソース・タグでサブ・セグメントを選択し，メイン・セグメントは M-seg インディペンデントとして選択．\\ \hline

（NR，R） & 第 1 ソース・タグでメイン・セグメントを選択し，サブ・セグメントは S-seg インディペンデントとして選択．選択したセグメントに空きがない場合，スワップして，第 1 ソース・タグでサブ・セグメントを選択し，メイン・セグメントは M-seg インディペンデントとして選択．\\ \hline

（R，R） & メインかつサブ・セグメント・インディペンデントとしてセグメントを選択． \\ \hline

\end{tabular}

\label{tab:cons\_algorithm\_subseg}

\end{table}

（NR，NR），（R，NR），（NR，R）の場合に関して，例を用いて以下で説明する．メイン・セグメント数を 4，サブ・セグメント数を 2 とし，命令 $p5 = p13 + p6$ をディスパッチする場合について例示する．第 1 ソース・タグが 13 で，第 2 ソース・タグが 6 である．

\begin{itemize}

\item （NR，NR）：第 1 ソース・タグでメイン・セグメントを，第 2 ソース・タグでサブ・セグメントを選択する．この場合，第 1 ソース・タグ 13（${\rm 1101\_2}$），第 2 ソース・タグ 6（${\rm 110\_2}$）より，（1，0）のセグメントを選択する． もし（1，0）に空きがない場合はスワップを行い，第 2 ソース・タグでメイン・セグメントを，第 1 ソース・タグでサブ・セグメントを選択する．この場合，（2，1）が選択される．なおも空きがない場合はストールする．

\item （R，NR）：スワップを行い，第 2 ソース・タグでメイン・セグメントを選択する．例の場合，第 2 ソース・タグ 6（${\rm 110\_2}$）より，メイン・セグメントは 2 となる．第 1 ソース・オペランドは既にレディであるため，S-seg インディペンデントである．従って，（2，0）または（2，1）のいずれかのセグメントを選択する．（2，0）と（2，1）のいずれも空きがない場合は，スワップをやめ，第 2 ソース・タグでサブ・セグメントを決定する．この場合，サブ・セグメントは 0 となる．第 1 ソース・オペランドは既にレディであるため，M-seg インディペンデントである．したがって（0，0），（1，0），（2，0），（3，0）のいずれかのセグメントが選択される．なおも空きがない場合はストールする．

\item （NR，R）：第 1 ソース・タグでメイン・セグメントを選択する．例の場合，第 1 ソース・タグ 13（${\rm 1101\_2}$）より，メイン・セグメントは 1 となる．第 2 ソース・オペランドは既にレディであるため，S-seg インディペンデントである．従って，（1，0）または（1，1）のいずれかのセグメントを選択する．（1，0）と（1，1）のいずれも空きがない場合は，スワップを行い，第 1 ソース・タグでサブ・セグメントを決定する．この場合，サブ・セグメントは 1 となる．第 2 ソース・オペランドは既にレディであるため，M-seg インディペンデントである．したがって（0，1），（1，1），（2，1），（3，1）のいずれかのセグメントが選択される．なおも空きがない場合はストールする．

\end{itemize}

\subsection{モードの切り替え}

SWITCH 方式では， AGGRESSIVE と CONSERVATIVE の 2 つのモードを，実行プログラムの ILP や MLP の量に応じて切り替えて使用する．ここで重要となるのは ILP や MLP の量の評価方法である．

本研究では， ILP の評価方法として Instructions Per Cycle（IPC）と Issue Stall Rate（ISR） という評価値が有効ではないかと考えた．また， MLP の評価方法としては，最終レベル・キャッシュ（LLC: last-level cache）の MPKI（misses per kilo instructions）が有効ではないかと考えた．それぞれに関して詳しく説明した後，切り替えアルゴリズムを説明する．

なお，評価の結果，ILP を評価する評価値としては，IPC と ISR のどちらも有効であるが，IPC のほうがより精度が高いことが分かったため，IPC を ILP の評価値として使用する．これらの評価は\refsec{eval\_threshold}で説明する．

\subsubsection{Instructions Per Cycle（IPC）}

IPC は，「サイクルあたりの平均コミット命令数」を表す指標であり，プロセッサの性能指標として一般的に使用される評価値である．IPC が高い場合，ILP は高いと判断される．

あらかじめ IPC にしきい値を設け，定期的にIPCを観測し，インターバルでの IPC がしきい値を上回った場合に ILP が高いと判断し，そうでなければ低いと判断する．

\subsubsection{Issue Stall Rate（ISR）}

ISR は，「インターバルの全サイクルのうち 1 命令も発行されないサイクルの割合」を表す指標である．ILP が高い場合，多くのサイクルで命令が発行されるため，ISR は低い値を示す．一方， ILP が低い場合には，命令が発行されないサイクルが一定の割合で発生するため，ISR は高くなる．

あらかじめISR にしきい値を設け，定期的にISRを観測し，インターバルでの ISR がしきい値を下回った場合に ILP が高いと判断し，そうでなければ低いと判断する．

\subsubsection{LLC MPKI}

LLC MPKI は LLC のキャッシュ・ミスの発生頻度を表す指標である．LLC MPKI があらかじめ定めたしきい値を上回った場合に MLP が高いと判断し，そうでなければ低いと判断する．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{switch}

\caption{SWITCH 方式におけるディスパッチ・エントリの決定回路}

\label{fig:switch}

\end{figure}

\subsubsection{切り替えアルゴリズム}

切り替えアルゴリズムは以下に示すとおりである．定期的に評価指標であるIPC（もしくはISR）と LLC MPKI を測定し，ILP および MLP の高低を判断する．ILP または MLP のいずれかが高いと判定された場合，次のインターバルを CONSERVATIVE モードで実行する．ILP と MLP がどちらも低いと判定された場合，次のインターバルを AGGRESSIVE モードで実行する．

SWITCH 方式におけるディスパッチするエントリの決定回路を\fig{switch}に示す．AGGRESSIVE と CONSERVATIVE の 2 つの選択アルゴリズムのうち，どちらを利用するかを SWITCH 回路が選択し，その結果に応じてセグメントが選択される．

+++++++++++++++++++++++++++++++++++++

src/eval

+++++++++++++++++++++++++++++++++++++

\chapter{評価}

\label{sec:eval}

本章では，提案手法の評価を行う．\refsec{eval\_env}で評価環境について説明し，\refsec{eval\_segIQ}で提案手法によるタグ比較回数と性能低下の評価を行う．\refsec{eval\_subseg}でサブ・セグメントに関する評価を説明した後，\refsec{eval\_threshold}で SWITCH 方式のパラメータに関する評価を行い，\refsec{eval\_ipc\_comp}でセグメント数に関する評価について説明する．

\section{評価環境}

\label{sec:eval\_env}

評価環境について説明する．性能やタグ比較回数を評価するために，SimpleScalar v.3.0a~\cite{ SimpleScalar } をベースに作成したシュミレータを使用した．評価で仮定したプロセッサ構成を\tab{base\_config}に示す．

提案手法の SWITCH 方式に関するパラメータを，\tab{switch\_config}に示す．これらのパラメータは，\refsec{eval\_threshold} で説明する評価に基づいて決定した最適なパラメータである．

測定ベンチマークには，SPEC CPU 2017 ベンチマークのうち，int 系 9 本と fp 系 9 本の計 18 本を使用した(gccとwrfは，現在のところ，シミュレータでは正しく動作しなかったため，除いている)．プログラムの入力には refspeed データ・セットを用いた．ベンチマークの測定区間は，プログラムの先頭から 16B 命令をスキップした後の 100M 命令である．

\begin{table}[htb]

\caption{プロセッサの基本構成}

\footnotesize

\center

\begin{tabular}{l|l} \hline \hline

Pipeline width & 8 instructions wide for each of \\

& fetch，decode，issue，and commit \\

Reorder buffer & 300 entries \\

IQ & 128 entries，w/ age matrix \\

Load/Store queue & 128 entries \\

Physical registers & 300（int） + 300（fp） \\

Branch prediction & 16KB Perceptron predictor~\cite{Jimenez2001} \\

& 2K-set 4-way BTB \\

& 10-cycle misprediction penalty \\

Function unit & 4 iALU，2 iMULT，\\

& 3 FPU，2 LSU \\

L1 D-cache & 32KB，8-way，64B line \\

& 2-cycle hit latency \\

L1 I-cache & 32KB，8-way，64B line \\

& 2-cycle hit latency \\

L2 cache & 2MB，16-way，64B line \\

& 12-cycle hit latency \\

Main memory & 300-cycle latency \\

& 8B/cycle bandwidth \\

Prefetch & stream-based，32-stream tracked， \\

& 16-line distance，2-line degree，\\

& prefetch to L2 cache \\ \hline

\end{tabular}

\label{tab:base\_config}

\end{table}

\begin{table}[tb]

\caption{提案手法の SWITCH 方式に関するパラメータ構成}

\footnotesize

\center

\begin{tabular}{l|l} \hline \hline

切り替えインターバル & 10K instructions \\

IPC しきい値 & 3.5 \\

LLC MPKI しきい値 & 2.0 \\ \hline

\end{tabular}

\label{tab:switch\_config}

\end{table}

\subsubsection{ベンチマークの分類}

提案手法は，プログラムの ILP や MLP に着目した制御を行う．そこで，SPEC CPU 2017 ベンチマークを ILP が高いベンチマーク，MLP が高いベンチマーク，いずれも低いベンチマークの 3 種類に分類する．ここて，ILP 及びMLP が高いベンチマークとは，次の条件を満たすベンチマークとした．

\begin{itemize}

\item high ILP：IPCが 3.5以上のベンチマーク

\item high MLP：LLC MPKI が2.0 以上のベンチマーク

\end{itemize}

分類結果を\tab{classification}に示す．また，以降に示す図において，ILP（青色） 及びMLP（赤色）の表記は，ILP もしくは MLP が高いベンチマークであることを表す．

\begin{table}[htb]

\caption{ベンチマークの分類}

\footnotesize

\center

\begin{tabular}{c|l} \hline \hline

分類 & ベンチマーク\\ \hline

high ILP & xz，bwaves，cactuBSSN，cam4，\\

& imagick，pop2，roms\\ \hline

high MLP & omnetpp，xalancbmk，lbm\\ \hline

low ILP and low MLP & exchange2，leela，deepsjeng，mcf\\

& perlbench，x264，fotonik3d，nab \\ \hline

\end{tabular}

\label{tab:classification}

\end{table}

\subsubsection{評価モデル}

評価モデルは以下の 4 種類である．

\begin{itemize}

\item BASE：セグメント化しない通常の IQ を使用するモデル

\item AGGRESSIVE：提案手法において常に AGGRESSIVE モードで実行するモデル

\item CONSERVATIVE：提案手法において常に CONSERVATIVE モードで実行するモデル

\item SWITCH：提案手法の SWITCH 方式を使用するモデル

\end{itemize}

\subsubsection{タグ比較回数の測定}

提案手法によるタグ比較回数の削減を評価するために，タグ比較の回数を測定した．ここで，タグ比較の回数とは，\textbf{ウェイクアップ時にレディでないオペランドの比較において，タグが一致しなかった数}とする．これは，電力消費に関する以下の理由による．

\begin{itemize}

\item すでにレディなオペランドの比較器は動作させない．比較器のプリチャージを抑制することにより，容易に停止させることができ，このとき電力を消費しない．

\item デスティネーション・タグは，最大で発行幅分送られてくるが，送られてこなかったデスティネーション・タグのタグ線につながっている比較器は動作しないとする．ここで「送られてこなかったタグ」とは，物理的には，プロセッサ内で使用されないタグ（偽のタグ）を定めそれを送信することとする．偽のタグとしてすべてのビットが0のタグを選択する．これにより，偽のタグの放送においては，タグ線はすべて$L$ となり，比較器のマッチ線はディスチャージされず，電力を消費しない．

\item タグが一致した比較器は，プリチャージされたマッチ線の電荷がディスチャージされないので，電力を消費しない．

\end{itemize}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{comp\_16\_1}

\caption{提案手法によるタグ比較回数（16，1）}

\label{fig:comp\_16\_1}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{ipc\_16\_1}

\caption{提案手法による性能低下（16，1）}

\label{fig:ipc\_16\_1}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{occupancy\_16\_1}

\caption{IQ の占有率（16，1）}

\label{fig:occupancy\_16\_1}

\end{figure}

\section{提案手法によるタグ比較回数削減と性能低下の評価}

\label{sec:eval\_segIQ}

提案手法によるタグ比較回数の削減と性能低下に関して評価を行う．サブ・セグメントを使用せず，セグメント数は 16 とした．このセグメント数は，提案手法による性能低下とタグ比較回数のバランスを考慮し，提案手法の特徴をよく評価できるパラメータであると考え選んだものである．

なお，以降の評価において，（メイン・セグメント数，サブ・セグメント数） の形式でセグメント数を表記する．また，サブ・セグメントを使用しない場合は，サブ・セグメント数は 1 と表記する．今回の場合は（16，1）となる．

\subsection{タグ比較回数の削減}

\fig{comp\_16\_1}に，提案手法の BASE モデルに対するタグ比較回数の割合をベンチマークごとに示す．GM は全ベンチマークでの幾何平均を表す．AGGRESSIVE と CONSERVATIVE のタグ比較回数削減に関しては、同図より，いずれのベンチマークにおいても，AGGRESSIVE の方がタグ比較回数が少ないことがわかる．その差は 平均で 20\%ポイント程度となっており，AGGRESSIVE モードのタグ比較回数を積極的に削減できるという性質が確認できる。

同図より SWITCH 方式では，平均で BASE モデルの 25\% 程度のタグ比較回数となっており，75\% の削減を達成している．

SWITCH 方式では，ILP や MLP の高いベンチマークにおいては CONSERVATIVE と同程度のタグ比較回数であるのに対して，そうでないベンチマークにおいては AGGRESSIVE に近いタグ比較回数となっていることがわかる．したがって，IQ の容量効率が重要でないベンチマークにおいては，AGGRESSIVE モードを選択して積極的にタグ比較回数の削減が行えていることがわかる．

\subsection{性能低下}

\fig{ipc\_16\_1}に，BASE に対する提案手法による性能低下をベンチマークごとに示す．同図より，SWITCH 方式による性能低下は最大で 3.5\% 程度であり，多くのベンチマークでは 0\% に近く性能はほとんど低下しないということが確認できる．

SWITCH 方式の有効性に関して述べる．同図より，ILP や MLP が高い xalancbmk や imagick，lbm などのベンチマークにおいて，AGGRESSIVE では大きく性能低下しているのに対して，CONSERVATIVE では性能低下が抑制されていることが分かる．そして SWITCH では，CONSERVATIVE と同程度の性能低下にとどまっている．従って，容量効率が性能にとって重要なプログラムにおいて，SWITCH 方式によって性能低下が抑制できていることが分かる．

\fig{occupancy\_16\_1}に各モデルでの IQ の占有率を示す．占有率とは，IQ の全エントリのうち1サイクルあたり使用されたエントリの割合であり，この値が BASE のそれに近いほど容量効率が低下していないことを示す．

同図より，AGGRESSIVE では BASE に対して占有率が大きく低下しているのに対して，CONSERVATIVE では占有率の低下がある程度抑制できていることが分かる．そして，ILP や MLP が高いベンチマークでは SWITCH 方式での占有率が CONSERVATIVE と同程度となっていることが分かる．このことからも，SWITCH 方式では，容量効率の性能に対する重要性に応じて適切にモードを選択し，容量効率の低下による性能低下を抑制できており，SWITCH 方式が有効であると言える．

\fig{ipc\_16\_1}より，いくつかのベンチマークでは性能が向上していることが分かる．この理由に関して説明する．一般にランダム・キュー方式の IQ には，命令がプログラム順に並んでいないため，最も優先して発行すべき命令の発行が遅れる可能性があるという欠点が存在する．ランダム・キューでは，命令の並びが年齢についてランダムになる一方，選択論理は，下のエントリほど優先して発行命令を選択するため，レディ命令が発行幅以上に存在する発行コンフリクトが生じた場合，性能に好影響を与えない命令の選択が生じる．

提案手法では IQ の容量効率が低下するため，IQ 内の命令数が少なくなり，結果的に発行コンフリクトが生じる確率が低下する．この結果，問題が生じにくくなり僅かに性能が向上する．

\fig{ipc\_16\_1}において，性能が向上するベンチマークの他に，ILP や MLP が高いにもかかわらず，AGGRESSIVE においても性能低下の小さいベンチマークがあることも分かる．こういったベンチマークにおいても，発行コンフリクトの緩和による性能向上が発生しているため，容量効率の低下による性能低下と中和されて，結果として性能低下が小さいと考えられる．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{comp\_8\_2}

\caption{提案手法によるタグ比較回数（8，2）}

\label{fig:comp\_8\_2}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{conservative\_sub\_seg}

\caption{サブ・セグメントを使用する CONSERVATIVE モードにおけるタグ比較回数}

\label{fig:conservative\_sub\_seg}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{ipc\_8\_2}

\caption{提案手法による性能低下（8，2）}

\label{fig:ipc\_8\_2}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{occupancy\_8\_2}

\caption{IQ の占有率（8，2）}

\label{fig:occupancy\_8\_2}

\end{figure}

\begin{table}[htb]

\caption{（16，1）と（8，2）の比較}

\footnotesize

\center

\begin{tabular}{cc|c|c|c} \hline \hline

& & タグ比較回数 & 性能低下（最大） & 性能低下（平均）\\\hline

& AGGRESSIVE & 16\% & 10.8\% & 0.6\% \\

（16，1） & CONSERVATIVE & 35\% & 2.2\% & 0.3\% \\ \

& SWITCH & 25\% & 3.8\% & -0.8\% \\ \hline

& AGGRESSIVE & 15\% & 3.7\% & -0.1\% \\

（8，2） & CONSERVATIVE & 21\% & 2.8\% & 0.3\% \\

& SWITCH & 18\% & 3.4\% & -0.4\% \\ \hline

\end{tabular}

\label{tab:subseg\_eval}

\end{table}

\section{サブ・セグメントに関する評価}

\label{sec:eval\_subseg}

サブ・セグメントを使用する場合の提案手法に関して評価する．メイン・セグメント数が 8，サブ・セグメント数が 2 の場合（（8，2）と表記する）に関して評価を行った．このセグメントの組み合わせは，（16，1） の場合とセグメントの総数が同じであり，比較の対象として適していると考え選んだものである．また，（8，2）の組み合わせは，\refsec{eval\_ipc\_comp}で説明するセグメントの分割数に関する評価において，性能と電力削減のバランスにおいて最適と判断された組み合わせである．

\subsection{タグ比較回数の削減}

\fig{comp\_8\_2}に，提案手法の BASE モデルに対するタグ比較回数の割合をベンチマークごとに示す．また，\tab{subseg\_eval}に，（16，1） と（8，2）の各評価モデルにおけるタグ比較回数の平均値，性能低下の最大値，性能低下の平均値を示す．

\tab{subseg\_eval} より，（16，1）と （8，2）の CONSERVATIVE のタグ比較回数を比較すると，（16，1）の場合が平均で 35\% であるのに対して，（8，2） では 21 \% 程度と，（8，2）のほうがより削減できていることが分かる．これは，サブ・セグメントを使用する場合，\refsec{two\_mode}で説明した CONSERVATIVE モードでのストールの回避を行った際にも，タグ比較の削減が可能となるためである．\fig{conservative\_sub\_seg}を用いて詳しく説明する．

図中に示す命令をディスパッチする場合を考える．サブ・セグメントを使用しない場合（図左側），第 1 ソース・タグ $p4$ によって決定されるセグメント（第 4 セグメント）に空きがなければ，CONSERVATIVE ではスワップを行い，セグメント・インディペンデントとしてディスパッチを行う．この場合，第 1 ソース・タグ $p4$ のタグ比較回数は削減されない．

一方で，サブ・セグメントを使用する場合（図右側），第 1 ソース・タグ$p4$によってメイン・セグメントが決定され，もし空きがなければ，スワップを行う．そして，第 1 ソース・タグ $p4$ によってサブ・セグメント番号が決定され，該当する番号のサブ・セグメントのいずれかに空きがあれば（図中の黄色で示したセグメント），そのセグメントにディスパッチする．この場合，第 1 ソース・タグ $p4$ の比較は，タグの下位ビットがサブ・セグメント番号と一致する場合のみ行われるため，その比較回数は 「1 / サブ・セグメント数」まで削減が可能となる．

以上で説明したように，サブ・セグメントを用いることにより，CONSERVATIVE モードでストールの回避を行った際にも，タグ比較の削減の機会が増加する．その結果サブ・セグメントを使用する場合，使用しない場合に比べて，CONSERVATIVE モードで高いタグ比較削減率を達成することができる．

最後に SWITCH 方式に関して評価する．（16，1） の場合と同様に容量効率の重要性に応じてモードの切り替えができており，IQ の容量効率が重要でないベンチマークにおいては AGGRESSIVE モードと同程度の削減率を達成できている．

\subsection{性能低下}

\fig{ipc\_8\_2}に，BASE に対する提案手法による性能低下をベンチマークごとに示す．同図より，SWITCH 方式による性能低下は最大で 4\% 程度であり，多くのベンチマークでは 0\% に近く性能はほとんど低下しないということが確認できる．

AGGRESSIVE モードでの性能低下率に関して考える．\tab{subseg\_eval} より，（8，2） では，（16，1）と比較して AGGRESSIVE モードでの性能低下率が低いことが分かる．これは，サブ・セグメントによって AGGRESSIVE モードでの容量効率の低下が抑制されているためであると考えられる．

\fig{occupancy\_8\_2} に，（8，2）の場合の IQ の占有率を示す．\fig{occupancy\_16\_1}と\fig{occupancy\_8\_2}の占有率を比較すると，（16，1）の場合は平均で 30\% 程度であった占有率が，（8，2）では平均で 40\% となっている．また，（16，1）の AGGRESSIVE モードにおいて特に性能低下の大きかった imagick に関して見てみると，（16，1）の AGGRESSIVE モードでは占有率が 35\% 程度で，性能低下が10\% であるのに対して，（8，2） の AGGRESSIVE モードでは占有率が 65\% 近くまで上昇しており，その結果性能低下が 3\% 程度となっている．

以上の考察から，セグメントの総数が同じである場合，サブ・セグメントを使用することによって，AGGRESSIVE モードでの容量効率の低下による性能低下を抑制できることがわかった．

最後に，SWITCH 方式の有効性に関して説明する．（8，2）の場合，サブ・セグメントが有効であり，AGGRESSIVE での大幅な性能低下が見られないため，（16，1）の場合と比較して SWITCH 方式の有効性は平均では高くない．しかし，imagick や lbm などのベンチマークにおいては AGGRESSIVE モードで発生する性能低下を抑制できている．また，CONSERVATIVE モードでのタグ比較削減率が高く，その結果 SWITCH 方式自体のタグ比較削減率も高くなっている．

\subsubsection{サブ・セグメントの評価のまとめ}

サブ・セグメントを使用すると，以下のメリットがあることがわかった．

\begin{itemize}

\item CONSERVATIVE モードでのタグ比較回数がより削減できる．

\item AGGRESSIVE モードにおける性能低下が改善される．

\end{itemize}

したがって，サブ・セグメントは有効な手法であると言える．

\section{SWITCH 方式のしきい値に関する評価}

\label{sec:eval\_threshold}

SWITCH 方式において，ILP と MLP の高低を判定するために使用するしきい値に関する評価を行う．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{switch\_IPC\_rate}

\caption{IPC を用いた SWITCH 方式の制御}

\label{fig:switch\_IPC\_rate}

\end{figure}

\subsection{ILP の評価値}

ILP を評価する値として IPC と ISR の評価を行う．評価の方針としては，まず IPC と ISR のしきい値に関して適当な値を求める．適当ななしきい値は，しきい値を変化させた場合に，ILP の高いベンチマークにおいて CONSERVATIVE モードで実行される割合を評価することによって決定する．

その後，それぞれの評価値を利用する場合の提案手法によるタグ比較削減と性能低下を比較し，IPC と ISR のどちらがより適した評価指標か評価する．

\subsubsection{IPC による制御}

ILP の評価指標として，IPC を使用する場合の評価を行った．本評価は，セグメント数を（16，1）として行った．また，ILP による SWITCH 方式の制御のみ行い，MLP による制御は行っていない．

\fig{switch\_IPC\_rate}に，IPC のしきい値を変化させた場合の，CONSERVATIVE モードで実行される割合を示す．この割合が大きいほど，ILP が高いと判断され多くのサイクルが CONSERVATIVE モードで実行されていることを表す．また，凡例の IPC=X は，ILP が高いと判定する IPC のしきい値をX とした場合を示している．また，avg は全ベンチマークの平均を，high ILP avg は ILP の高いベンチマークでの平均を表している．

当図より，IPC のしきい値が高くなるほど，CONSERVATIVE モードで実行される割合が小さくなっていることが分かる．これは，ILP が高いと判定される基準が厳しくなるためである．ILP の高いベンチマークに関して見ると，多くのベンチマークにおいて，しきい値が 3.5 の場合は CONSERVATIVE モードの割合が大きく，しきい値が 4.0 になると CONSERVATIVE モードの割合が小さくなっていることが分かる．high ILP avg を見ると，しきい値が 3.5 の場合は CONSERVATIVE モードの割合は 90\% 程度であるが，しきい値が 4.0 の場合は CONSERVATIVE モードの割合は 60\% 程度と小さくなる．

ILP が高い場合は CONSERVATIVE モードで実行することが望ましい．従って，IPC のしきい値は，3.5 程度が適当であるといえる．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{switch\_ISR\_rate}

\caption{ISR を用いた SWITCH 方式の制御}

\label{fig:switch\_ISR\_rate}

\end{figure}

\subsubsection{ISR による制御}

ILP の評価指標として，ISR を使用する場合の評価を行った．評価は IPC の場合と同様にセグメント数を（16，1）として行った．また，ILP による SWITCH 方式の制御のみ行い，MLP による制御は行っていない．

\fig{switch\_ISR\_rate}に，ISR のしきい値を変化させた場合の，CONSERVATIVE モードで実行される割合を示す．凡例の ISR=X は，ILP が高いと判定する ISR のしきい値を X とした場合を表す．

同図より，ISR のしきい値が低くなるほど，CONSERVATIVE モードで実行される割合が小さくなっていることが分かる．これは，ILP が高いと判定される基準が厳しくなるためである．high ILP avg を見ると，しきい値が 5 の場合は CONSERVATIVE モードの割合は 60\% 程度であるが，しきい値が 10 の場合は CONSERVATIVE モードの割合は 90\% 程度と大きくなる．

ILP が高い場合は CONSERVATIVE モードで実行することが望ましい．従って，ISR のしきい値は，10 程度が適当であるといえる．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{switch\_ILP\_performance}

\caption{ILP による制御を行った SWITCH 方式による性能低下}

\label{fig:switch\_ILP\_performance}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{switch\_ILP\_comp}

\caption{ILP による制御を行った SWITCH 方式によるタグ比較回数}

\label{fig:switch\_ILP\_comp}

\end{figure}

\subsubsection{IPC と ISR の比較}

\fig{switch\_ILP\_performance}に，IPC と ISR を用いて制御を行う SWITCH 方式での性能低下を示す．MLP による制御は行っていない．同図より，ILP が高いベンチマークにおいては，IPC と ISR いずれの指標においても正しく評価できており，結果として SWITCH による性能低下が CONSERVATIVE と同程度に抑制できていることがわかる．また，ILP が高くないベンチマークにおいては，IPC と ISR による制御において大きな差はみられない．平均を見ても，IPC と ISR は同程度であると言える．

\fig{switch\_ILP\_comp}に，IPC と ISR を用いて制御を行う SWITCH 方式でのタグ比較回数を示す．タグ比較回数も，性能低下と同様に IPC と ISR で大きな差は見られないが，exchange2 と fotonik3d に関しては，IPC を用いた制御のほうが ISR を用いた制御よりもタグ比較回数が少ないことが分かる．この理由は，この 2 つのベンチマークについては，IPC を用いた制御で ILP が低いと判定されているのに対して，ISR を用いた制御では ILP が高いと判定されているためである．

exchange2 及び fotonik3d は提案手法によって性能低下を起こさないベンチマークであるため，ILP が高いと判定されて CONSERVATIVE モードで実行されることは望ましくない．従って，これらのベンチマークを ILP が低いと判定できる IPC による制御のほうが適していると考えられる．

以上の評価から，ISR による制御は一部の ILP が低いベンチマークを ILP が高いと判定してしまうため，IPC を 用いた制御のほうがより適していると言える．SWITCH 方式における ILP の判定は，IPC を用いることとし，ILP が高いとする IPC のしきい値は 3.5 とする．

\subsection{MLPの評価値}

MLP を評価する値として LLC MPKI の評価を行う．評価の方針としては，LLC MPKI のしきい値に関して適当な値を求める．適当なしきい値は，しきい値を変化させた場合に，MLP の高いベンチマークにおいて，CONSERVATIVE モードで実行される割合を評価することによって決定する．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{switch\_MPKI\_rate}

\caption{LLC MPKI による SWITCH 方式の制御}

\label{fig:switch\_MPKI\_rate}

\end{figure}

\subsubsection{LLC MPKI による制御}

本評価は，セグメント数を（16，1）として行った．また，MLP による SWITCH 方式の制御のみ行い，ILP による制御は行っていない．

\fig{switch\_MPKI\_rate}に，LLC MPKI のしきい値を変化させた場合の，CONSERVATIVE モードで実行される割合を示す．凡例の MPKI=X は，MLP が高いと判定する LLC MPKI のしきい値を X とした場合を表す．また，avg は全ベンチマークの平均を，high MLP avg は MLP の高いベンチマークでの平均を示している．

当図より，LLC MPKI しきい値が高くなるほど，CONSERVATIVE で実行される割合が小さくなっていることが分かる．これは，LLC MPKI が高いと判定される基準が厳しくなるためである．

MLP の高いベンチマークに関して考える．omnetpp と lbm においては，しきい値が 2 以下場合は CONSERVATIVE モードの割合が大きいが，しきい値が 3 以上 になると CONSERVATIVE モードの割合が急激に小さくなっていることが分かる．xalancbmk に関しては，MLP の高いベンチマークの中でも特にメモリ・インテンシブなベンチマーク（LLC MPKI が 10 程度）であるため，しきい値を 5 まで増加させても CONSERVATIVE の割合は大きく変化しない．

MLP が高い場合は CONSERVATIVE モードで実行することが望ましい．従って，LLC MPKI のしきい値は，2 程度が適当であるといえる．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{moderate\_16\_1}

\caption{SWITCH 方式におけるモードの割合（16，1）}

\label{fig:moderate\_16\_1}

\end{figure}

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{moderate\_8\_2}

\caption{SWITCH 方式におけるモードの割合（8，2）}

\label{fig:moderate\_8\_2}

\end{figure}

\subsection{IPC と LLC MPKI を用いた制御に関する評価}

ILP と MLP による制御を同時に行った場合に関して評価を行う．SWITCH 方式において，上記で決定したしきい値を用いて制御を行った際の，AGGRESSIVE モードと CONSERVATIVE モードで実行される割合を\fig{moderate\_16\_1}に示す．セグメント数は（16，1）である．

同図より，ILP 及び MLP の高いベンチマークは CONSERVATIVE モードの割合が多く，その他のベンチマークにおいては AGGRESSIVE モードの割合が高いことがわかる．したがって，容量効率の重要性に応じて適切なモードを選択できていると言える．

セグメント数が異なる場合でも同様の制御が行えるか確かめるため，（8，2）においても同様の測定を行った．結果を\fig{moderate\_8\_2}に示す．当図より （8，2） の場合においても，ILP または MLP が高いベンチマークは CONSERVATIVE モードの割合が高く，そうでないベンチマークでは AGGRESSIVE モードの割合が高いことがわかる．したがって，（16，1）で決定した IPC や LLC MPKI のしきい値は，異なるセグメント数であっても有効であるといえる．

\begin{figure}[htb]

\centering

\includegraphics[keepaspectratio, scale=.8]{segment\_num}

\caption{セグメント数の違いによるタグ比較回数と性能低下の散布図}

\label{fig:segment\_num}

\end{figure}

\section{セグメントの分割数に関する評価}

\label{sec:eval\_ipc\_comp}

セグメントの分割数を変化させて提案手法を評価し，最適であるセグメントの分割数を決定する．評価の基準としては，提案手法において次の条件を満たすセグメント数の組み合わせを最適と判断する．

\begin{itemize}

\item 性能低下が全ベンチマークで 5\% 以下

\item タグ比較回数の平均が最も少ない

\end{itemize}

\fig{segment\_num}に，メイン・セグメント数とサブ・セグメント数を変化させた場合の，SWITCH 方式での性能低下とタグ比較回数の散布図を示す．図中の各点にはメイン・セグメント数とサブ・セグメント数を（メイン・セグメント数，サブ・セグメント数）という形式で付与している．横軸は最も性能低下が大きかったベンチマークにおける性能低下を示し，縦軸は全ベンチマークでの平均のタグ比較回数を示す．

同図より，セグメントの総数が増えると，タグ比較回数はより削減されるが，同時に性能低下も大きくなることがわかる．特に，（8，4）や（16，2）など，セグメントの総数が 32 を超えると，性能低下率が急激に増加している．

\fig{segment\_num}において性能低下が 5\% より小さいセグメント数の組み合わせのうち，最もタグ比較回数の少ない（8，2）を最適が最適な組み合わせである．このときのタグ比較回数は 18\%（82\% 削減）となる．

また，サブ・セグメントを使用しない場合に最適な組み合わせは（16，1）であり，タグ比較回数は25\%（75\% 削減） となっている．

+++++++++++++++++++++++++++++++++++++

src/summary

+++++++++++++++++++++++++++++++++++++

\chapter{まとめ}

\label{sec:summary}

LSIの微細化の進展に伴って，経年劣化が加速し摩耗故障が増加する問題が深刻になっている．この故障は，デバイスの温度に関して指数関数的に加速するため，チップ内のホット・スポットの解消が求められている．

発行キューはこのホット・スポットの 1 つとして知られている．この主な原因はウェイクアップ時の多数のタグ比較である．本論文では，ウェイクアップ時のタグ比較回数を削減するために，発行キューをセグメント化し，それを最大限活用するいくつかの手法を提案した．

提案手法には発行キューの容量効率が低下するという問題が存在する．この問題点に対して，本論文ではさらに，異なる 2 つのディスパッチ・アルゴリズムを，性能についての容量効率の重要性に応じて切り替えて使用することにより，容量効率の低下による性能低下を抑制する手法を提案した．

提案手法を SPEC CPU 2017 を使って評価したところ，性能低下を最大でも 5\% 以下に抑えつつ，タグ比較回数を 82\% 削減できることを確認した．

+++++++++++++++++++++++++++++++++++++

src/publication

+++++++++++++++++++++++++++++++++++++

\chapter\*{発表実績}

\addcontentsline{toc}{chapter}{発表実績}

\begin{itemize}

\item 森健一郎, 安藤秀樹, ``容量効率を意識したソース・タグ値に基づくセグメント化による発行キューのエネルギー削減'', 情報処理学会研究報告, Vol.2020-ARC-241, No.3, pp.1-12, 2020年7月．

\end{itemize}

+++++++++++++++++++++++++++++++++++++

src/acknowledgement

+++++++++++++++++++++++++++++++++++++

\chapter\*{謝辞}

\addcontentsline{toc}{chapter}{謝辞}

本研究を進めるにあたり，多大なる御指導と御鞭撻を賜わりました名古屋大学大学院工学研究科 情報・通信工学専攻 安藤秀樹教授に心より感謝いたします．また，本研究の遂行を支えてくださいました，名古屋大学大学院工学研究科情報・通信工学専攻安藤研究室の諸氏に深く感謝します．