Vol.2012-ARC-200 No.7
Vol.2012-OS-121 No.7
2012/5/7

AnT におけるSH-4向けサーバプログラム間通信機構の高速化手法の評価

鶴谷 昌弘[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マイクロカーネル構造の ***AnT*** オペレーティングシステムは，Pentium4 プロセッサ上で走行し，高速なサーバプログラム間通信機構を有する．そこで，代表的な組み込みプロセッサの 1 つである SH-4 プロセッサへ移植し，サーバプログラム間通信機構の性能を評価した．しかし，単に移植しただけでは，高性能を期待できない．これを解決するため，SH-4 向けのサーバプログラム間通信機構の高速化手法を提案する．本稿では，SH-4 向けサーバプログラム間通信機構の高速化手法について述べ，高速化の効果について評価する．

Evaluation for Fast Method of Inter Server Program Communication on *AnT* for SH-4

Masahiro TSURUYA[1]　Toshihiro YAMAUCHI[1]　Hideo TANIGUCHI[1]

Abstract: ***AnT*** is an operating system based on microkernel architecture. ***AnT*** has a fast inter-server program communication (ISPC) mechanism, and works on Pentium4 processor. ***AnT*** had ported to SH-4 processor that one of the representative embedded processors, and had evaluated performance of ISPC. However, ISPC is required higher performance than that of ***AnT*** simply porting. Therefore, fast method of ISPC for SH-4 is proposed in this paper. This paper shows the proposed method, and evaluations of it.

1. はじめに

高い適応性と堅牢性を実現する OS のプログラム構造として，マイクロカーネル構造がある [1]–[3]．マイクロカーネル構造は，例外処理や割込処理といった最小限の OS 機能をカーネルとして実現し，ファイル管理やデバイスドライバなどの OS 機能をプロセスとして実現するプログラム構造である．つまり，多くの OS 機能は，カーネル外にプロセス（以降，OS サーバと呼ぶ）として実現する．これにより，機能の追加や削除を容易にできる．また，OS サーバごとに機能を分担させることにより，プログラムの暴走によるシステム全体の破壊を防止できる．しかし，マイクロカーネル構造では，OS サーバ間の通信が頻発するため，Linux のようなモノリシックカーネル構造の OS と比べ，性能が低下しやすい．そこで，この性能低下を抑制する機構が必要である．

我々は，マイクロカーネル構造を有する ***AnT*** オペレーティングシステム（An operating system with Adaptability and Toughness）（以降，***AnT*** と略す）を開発している [4]．***AnT*** は，高速なサーバプログラム間通信機構を持ち，Pentium4 プロセッサ（以降，Pentium4 と略す）上で走行する．

AnT の適応域を拡大するため，代表的な組み込みプロセッサの 1 つである SH-4 プロセッサ（以降，SH-4 と略す）へ ***AnT*** を移植し [5]，サーバプログラム間通信機構の性能を評価した．しかし，単に移植しただけでは，***AnT***

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

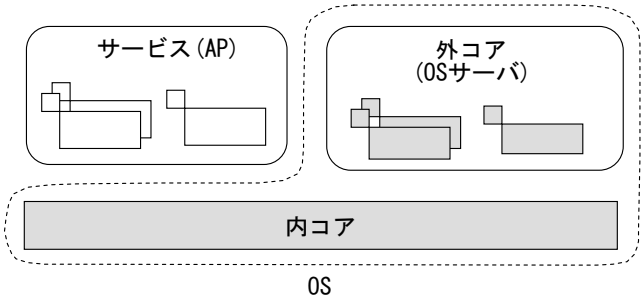


図 1 ***AnT*** の基本構造

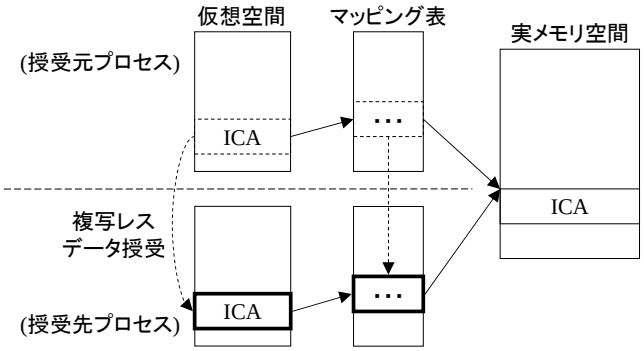


図 2 複写レスデータ授受の様子

のサーバプログラム間通信機構は Pentium4 用に設計されており，SH-4 の特徴は生かされていないため，高性能を期待できない．これを解決するため，SH-4 向けのサーバプログラム間通信機構の高速化手法を提案する．提案する高速化手法は，Pentium4 と SH-4 のアーキテクチャの差異に着目し，SH-4 のアーキテクチャの特徴を生かすものである．また，提案手法を評価し，高速化の効果を明らかにする．

2. *AnT* オペレーティングシステム

2.1 プログラム構造

AnT はマイクロカーネル構造を有するオペレーティングシステムである．***AnT*** のプログラムは，OS とサービスからなる．この様子を**図 1** に示す．OS は，カーネル（内コア）とプロセス（OS サーバ）として動作する外コアからなる．内コアは，最小のシステムの動作を保証するプログラム部分である．外コアは，システムの利用形態に適応するために必須なプログラム部分である．サービスは，サービスを提供するプログラム部分である．

2.2 複写レスデータ授受

プロセス間の通信を高速化するため，コア間通信データ域 (ICA: Inter-core Communication Area) を利用した複写レスでのデータ授受機能がある．ICA の特徴として以下の 3 つがある．

(1) ページ (4KB) を単位とし，n ページ分の領域の確保と解放

(2) 確保した領域 (n ページ) の実メモリ連続の保証

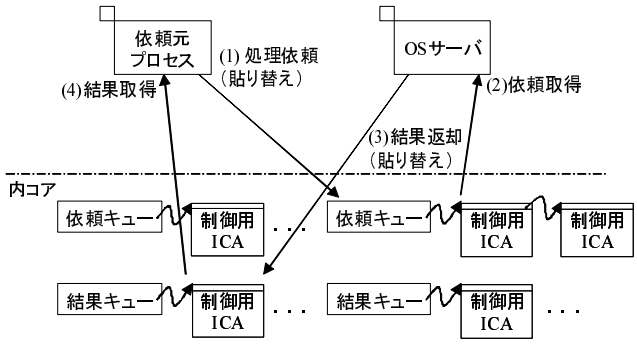


図 3 サーバプログラム間通信の基本機構

(3) 2 仮想空間での領域の貼り替え

ICA は，ページを最小単位として管理される領域であり，ICA へのアクセスは，プロセスごとの仮想空間のマッピング表を通して行われる．ここで，マッピング表への書き込みを貼り付けと呼び，マッピング表からの削除を剥がしと呼ぶ．また，貼り替えとは，剥がし&貼り付けを意味する．プロセス間の複写レスでのデータ授受の様子を**図 2** に示す．ICA を利用したプロセス間でのデータ授受は，授受するデータを格納した ICA をデータ授受元プロセスの仮想空間から剥がし，データ授受先プロセスの仮想空間へ貼り付けることで行われる．

2.3 サーバプログラム間通信機構 [4]

2.3.1 基本機構

サーバプログラム間通信の基本機構を**図 3** に示す．ICA を利用することにより，プロセス間の複写レスデータ授受を実現している．具体的には，OS サーバへ渡す引数や通信制御の情報（以降，依頼情報）を制御用の ICA（以降，制御用 ICA）に格納し，扱うデータをデータ用の ICA（以降，データ用 ICA）に格納する．内コアは，各プロセスごとに通信のための依頼キューと結果キューを持つ．基本的な通信の流れを以下に述べる．

(1) 依頼元プロセスが処理依頼を行うと，内コアは OS サーバの依頼キューに依頼情報を格納した制御用 ICA を登録し，OS サーバへ制御用 ICA を貼り替える．

(2) OS サーバは，依頼キューから依頼情報を格納した制御用 ICA を取得し処理を実行する．

(3) OS サーバが結果返却を行うと，内コアは依頼元プロセスの結果キューに結果情報を格納した制御用 ICA を登録し，依頼元プロセスへ制御用 ICA を貼り替える．

(4) 依頼元プロセスは，結果キューから結果情報を格納した制御用 ICA を取得し処理を終了する．

また，同期型と非同期型の通信インタフェースを同様な形式で提供し，両インタフェースを選択して利用できる．

2.3.2 多段依頼と直接返却

マイクロカーネル OS では，OS サーバを多段に経由して処理依頼が発行される．これを多段依頼と名付ける．こ

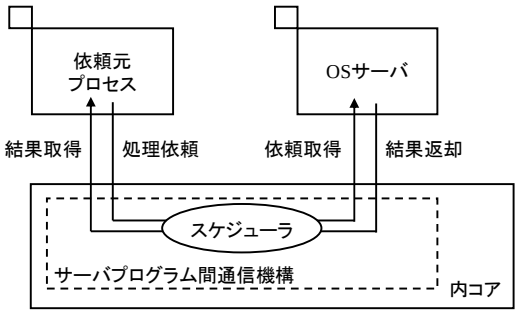


図 4 基本処理の測定の様子

表 1 測定環境

	SH-4	Pentium4
動作周波数	240MHz	2.8GHz
パイプライン段数	5 段	20 段
メインメモリ	32MB	256MB
命令キャッシュ	16KB	-
オペランドキャッシュ	32KB	-
L1 キャッシュ	-	16KB
L2 キャッシュ	-	256KB

のとき，制御用 ICA を処理依頼のたびに確保すると処理オーバヘッドが大きい．そこで，1 つの制御用 ICA を持ち回り，依頼情報を積み重ねることでオーバヘッドを抑制する．また，積み重ねられた情報を基に依頼元プロセスへ結果を渡すことで，結果返却を実現する．

多段依頼された処理は，積み重ねられた依頼情報を基に中継した OS サーバを経由した逐次的な返却が行われる．しかし，必ずしも逐次的な返却を行う必要がない場合は，依頼元のプロセスへ直接に返却することにより，処理を高速化する．具体的には，制御用 ICA に flag を設け，返却の要否を設定できることとした．結果返却時に内コアが flag を確認し，返却が必要な依頼元のプロセスに直接返却を行う．

3. 高速化手法

3.1 考え方

3.1.1 高速化前のサーバプログラム間通信性能

サーバプログラム間通信機構は，同期の処理依頼と結果返却，および非同期の処理依頼と結果返却という 4 つの基本処理からなる．また，それぞれの処理について，データ用 ICA の授受を行う場合と行わない場合がある．さらに，基本処理は，以下の部分処理からなる．

(1) システムコール発行
(2) ICA 貼り付け
(3) ICA 剥がし
(4) ICA への初アクセス
(5) プロセス切換

ここで，システムコール発行は，プロセスから内コアへの処理移行と内コアからプロセスへの処理の戻りを合わせた

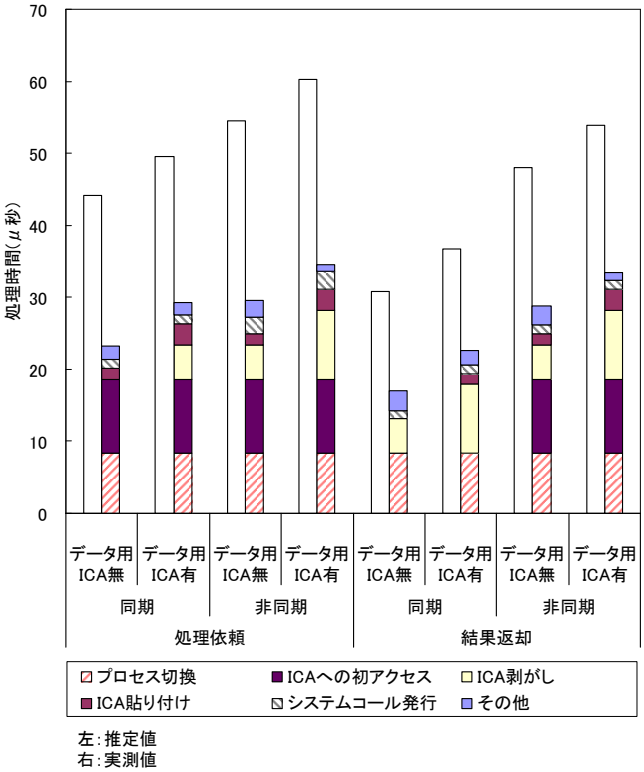


図 5 高速化前の基本処理の処理時間

処理である．また，ICA への初アクセスは，そのプロセスに ICA が貼り付けられた後，初めてアクセスした際に発生する処理である．

基本処理の測定の様子を**図 4** に示す．基本処理の測定は，依頼元プロセスと OS サーバで通信を行い，各処理時間を測定する．測定環境を**表 1** に示す．測定において，処理依頼は，OS サーバへ渡す引数および戻り値を無しとし，データ用 ICA のサイズを 4KB にした．また，通信機構のオーバヘッドを明確化するため，OS サーバでは通信に関連する処理以外の処理は行わない．

基本処理の処理時間を**図 5** に示す．ここで，推定値は，Pentium4 の動作周波数（2.8GHz）と SH-4 の動作周波数（240MHz）の比（約 11.7 倍）を基に，Pentium4 上での処理時間 [4] を 11.7 倍した値である．図 5 より，実測値は推定値より短いことがわかる．これは，基本処理では，走行モードの変更 [6] や仮想空間の切り換えが発生し，パイプライン機能やメモリキャッシュ機能が有効に働かないためである．

3.1.2 考察

図 5 に示したように，SH-4 上でのサーバプログラム間通信機構の基本処理の処理時間は，Pentium4 上での処理時間からの推定値より短く，高性能といえる．しかし，単に移植しただけであるため，サーバプログラム間通信機構は Pentium4 用に設計されており，SH-4 の特徴を生かしていない．このため，SH-4 の特徴を生かせば，さらなる高速化が期待できる．

図 5 より，SH-4 上での基本処理の処理時間の約 90%は，ICA 貼り付け，ICA 剥がし，ICA への初アクセス，およびプロセス切換の処理時間である．ここで，ICA 貼り付けと ICA 剥がしは，通信中に頻発する．このため，ICA 貼り付けと ICA 剥がしの処理回数を削減することにより，高速化が期待できる．また，ICA への初アクセスは，TLB（Translation Lookaside Buffer）ミスによるオーバヘッドであるため，TLB ミスの発生を抑制することにより，高速化が期待できる．一方，プロセス切換は，SH-4 の特徴を生かした処理となっているため，高速化は期待できない．

3.2 SH-4 の MMU の特徴

SH-4 の MMU（Memory Management Unit）の特徴を以下に示す．

(特徴 1) TLB ミス発生時に例外が発生し，ソフトウェアへ処理が移行する．このため，TLB へのアドレス変換情報の登録をソフトウェアで行え，TLB ミスを削減できるような設定が可能である．

(特徴 2) ソフトウェアで事前に TLB へのアドレス変換情報の登録を行うことにより，TLB ミスの発生を防ぐことができる．

(特徴 3) 一つの仮想空間内において，ページサイズを 1KB，4KB，64KB，および 1MB から選択でき，各ページサイズを混在して利用できる．

(特徴 4) TLB は 64 エントリで構成され，TLB へアドレス変換情報を登録する際，どのエントリを利用するか指定できる．なお，指定しない場合，ハードウェアのランダムカウンタの値により決定される．

(特徴 5) 多重仮想記憶を利用する場合，TLB のエントリ情報として各仮想空間の識別子を保持する．このため，仮想空間切り換えの際に必ずしも TLB をフラッシュする必要がない．

一方，SH-4 上に移植した ***AnT*** は，ページサイズ 4KB 固定で利用している．また，TLB へアドレス変換情報の登録を行う際，エントリの指定を行っていない．ただし，仮想空間の切り換えに（特徴 5）を利用している．

3.3 方針

以下の方針で高速化する．

(1) 基本方式を変更しない．これは，サーバプログラム間通信機構の利用インタフェースの変更を防ぐためである．

(2) SH-4 の MMU の特徴を生かす．これにより，高速化が期待できる．

(3) サーバプログラム間通信機構の利用形態を考慮する．具体的には，OS サーバ数と ICA を利用する AP プロセス数を考慮し，それらを制限した形で制御する．これにより，ICA の管理を簡素化し高速化が期待できる．

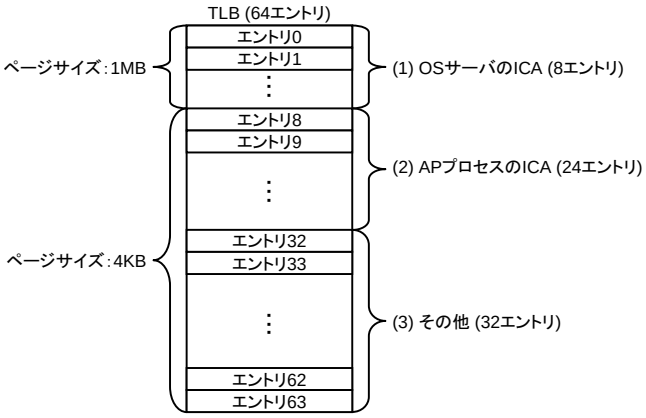


図 6 TLB エントリの分割管理

3.4 実現方式

TLB ミスと ICA の管理オーバヘッドを削減することにより，サーバプログラム間通信機構を高速化する．まず，以下のように ICA を管理する．

(1) 通信中に発生する TLB ミスは，主に ICA へのアクセス時に発生する．このため，ICA のアドレス変換情報を TLB に常駐させる．

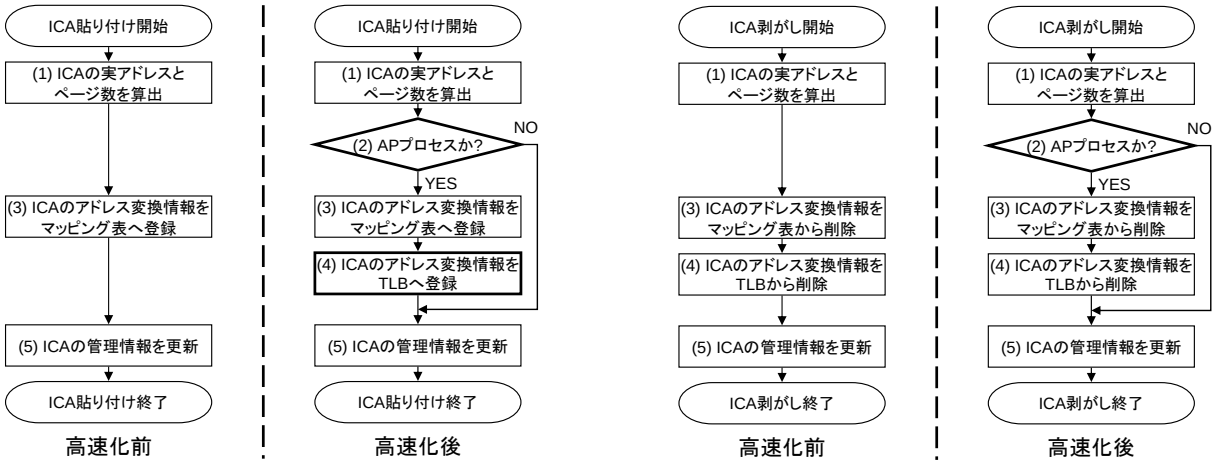
(2) OS サーバは，AP プロセスに比べ，処理の信頼性が高く，通信回数が多く，利用する ICA サイズも大きい．そこで，OS サーバは，全 ICA を常に読み書き可能とする．一方，AP プロセスが利用できる ICA のサイズを制限し，ICA の管理を簡素化する．

上記を実現するため，3.2 節で述べた SH-4 の MMU の特徴を利用する．（特徴 1）と（特徴 2）を利用し，ICA のアドレス変換情報を TLB に常駐させる．具体的には，ICA 貼り付け処理において，TLB へアドレス変換情報を登録する．また，（特徴 3）と（特徴 4）を利用し，TLB の全 64 エントリを利用目的別に分割して管理する．TLB エントリの分割管理の様子を**図 6** に示し，以下で説明する．

(1) OS サーバが利用する ICA は，ページサイズを 1MB とし，8 エントリを割り当て，各 OS サーバは 1 エントリを利用するものとする．この結果，OS サーバ数は最大 8 個，各 OS サーバで利用可能な ICA のサイズは最大 1MB になる．

(2) AP プロセスが利用する ICA は，ページサイズを 4KB とし，24 エントリを割り当て，各 AP プロセスは 4 エントリを利用するものとする．この結果，ICA を利用する AP プロセス数は最大 6 個，各 AP プロセスで利用可能な ICA のサイズは最大 16KB になる．また，OS サーバと異なり，自身に貼り付けられた ICA 以外はアクセスできないため，AP プロセスから ICA を保護できる．

(3) 上記（1）（2）以外は，ページサイズを 4KB とし，32 エントリを割り当てる．この際，TLB エントリの利用状態を管理し，TLB ミスを抑制する．具体的には，TLB へのアドレス変換情報の登録を行う際，利用されていないエ



(A) ICA 貼り付けの処理流れ　　(B) ICA 剥がしの処理流れ

図 7 ICA 貼り付けと ICA 剥がしの処理流れ

表 2 部分処理の処理時間

処理内容	高速化前	高速化後	
		OS サーバ	AP プロセス
ICA 貼り付け	1.5 μ秒	1.5 μ秒	5.8 μ秒
ICA 剥がし	4.8 μ秒	1.8 μ秒	5.7 μ秒
システムコール発行	1.2 μ秒	1.2 μ秒	1.2 μ秒
ICA への初アクセス	10.3 μ秒	0.5 μ秒	5.1 μ秒
プロセス切換	8.3 μ秒	5.8 μ秒	5.8 μ秒

ントリを指定する．全エントリが利用されている場合は，ハードウェアのランダムカウンタを用いてエントリを決定する．これにより，全エントリをランダムに利用するよりも TLB ミスの回数を削減できる．

上記の管理により，ICA 貼り付けと ICA 剥がしの処理を変更する．各処理の流れを**図 7** に示す．高速化後は，AP プロセスと OS サーバで ICA の管理方法が異なるため，AP プロセスか否かを判定する処理（2）を追加する．AP プロセスの場合，ICA 貼り付け処理において，TLB へのアドレス変換情報を登録する処理（4）を行う．一方，OS サーバの場合，ICA のアドレス変換情報は常に TLB へ登録されているため，ICA のアドレス変換情報をマッピング表へ登録や削除する処理（3）（4）を行わない．

4. 評価

4.1 基本性能

各部分処理の処理時間を**表 2** に示す．表 2 より，以下のことがわかる．

(1) OS サーバの高速化後の ICA 剥がし処理時間は，高速化前と比較し，約 3 μ秒 (62%) 減少している．これは，高速化後では，ICA のアドレス変換情報を TLB エントリから削除しないためである．

(2) OS サーバの高速化後の ICA への初アクセス処理時間は，高速化前と比較し，約 10 μ秒 (95%) 減少している．これは，高速化後では，ICA のアドレス変換情報は TLB に常駐しているため，ICA への初アクセスにおいて，TLB ミスが発生しないためである．

(3) AP プロセスの高速化後の ICA 貼り付け処理時間は，高速化前と比較し，約 4 μ秒 (386%) 増加している．これは，高速化後では，ICA のアドレス変換情報を TLB へ登録する処理を追加したためである．

(4) AP プロセスの高速化後の ICA 剥がし処理時間は，高速化前と比較し，約 1 μ秒 (16%) 増加している．これは，TLB エントリの分割管理により，ICA のアドレス変換情報を TLB から削除する処理において，TLB エントリの管理オーバヘッドが発生するためである．

(5) AP プロセスの高速化後の ICA への初アクセス処理時間は，高速化前と比較し，約 5 μ秒 (50%) 減少している．しかし，OS サーバの高速化後の ICA への初アクセス処理時間と比較すると，約 5 μ秒長い．これは，ICA のアドレス変換情報を TLB へ登録した後，対応するページに初めて書き込みを行った際，例外が発生するためである．

(6) 高速化後のプロセス切換の処理時間は，高速化前と比較し，約 2.5 μ秒 (30%) 減少している．これは，TLB エントリの分割管理により，TLB ミスの回数が減少したためである．

次に，4 つの基本処理の処理時間を**図 8** に示す．なお，AP プロセスの処理依頼と結果返却は，同期型のシステムコールインタフェースであるため，同期型の値のみである．図 8 より，以下のことがわかる．

(1) 高速化により，処理依頼時間は約 10〜17 μ秒 (48〜55%)，結果返却時間は約 5〜17 μ秒 (24〜53%) 減少した．これは，OS サーバの ICA 貼り付け，OS サーバの ICA 剥がし，ICA への初アクセス，およびプロセス切換の処理時間が減少したためである．

(2) 高速化後のデータ用 ICA 有の同期処理依頼と同期結果返却は，AP プロセスと OS サーバで処理時間が異なる．これは，ICA 貼り付け処理時間と ICA 剥がし処理時間は，

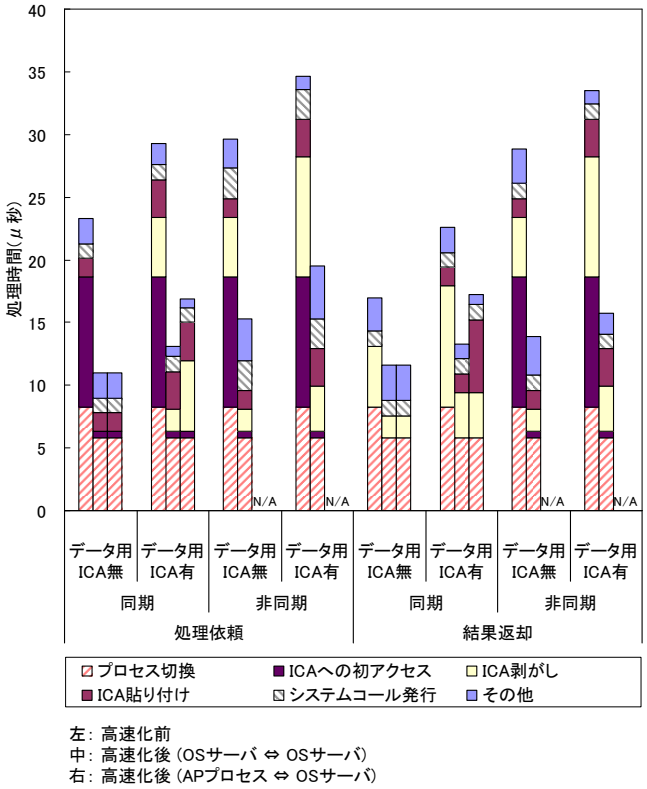


図 8 基本処理の処理時間

OS サーバと AP プロセスで異なるためである．

なお，図 8 のデータ用 ICA 有の場合，データ用 ICA へのアクセスは発生していない．これは，測定時に OS サーバは通信に関連する処理以外の処理を行なっていないためである．一方，実サービスにおいては，データ用 ICA へのアクセスが発生する．このとき，データ用 ICA への初アクセスの際，高速化前は TLB ミスが発生するのに対し，高速化後は発生しない．

4.2 直接返却

直接返却を利用した場合の処理時間について評価する．OS サーバを n 段介した処理依頼において，AP プロセスが処理依頼を発行してから結果が返ってくるまでの処理時間を**図 9** に示す．図 9 より，段数に関係なく，逐次返却と直接返却の時間を約 40%短縮できることがわかる．

5. 関連研究

5.1 superpage

文献 [7] は，superpage と呼ばれる大きいサイズのページを使用することにより，システムの性能向上を実現している．具体的には，メモリ使用量の増加により，TLB エントリが不足すると予測された場合，複数の通常サイズのページをまとめ，superpage へ変換する．これにより，TLB ミスの発生を抑制している．これに対し，提案した高速化手法では，利用目的にあわせ，ページサイズを選択している．具体的には，OS サーバが利用する ICA のページサイズを

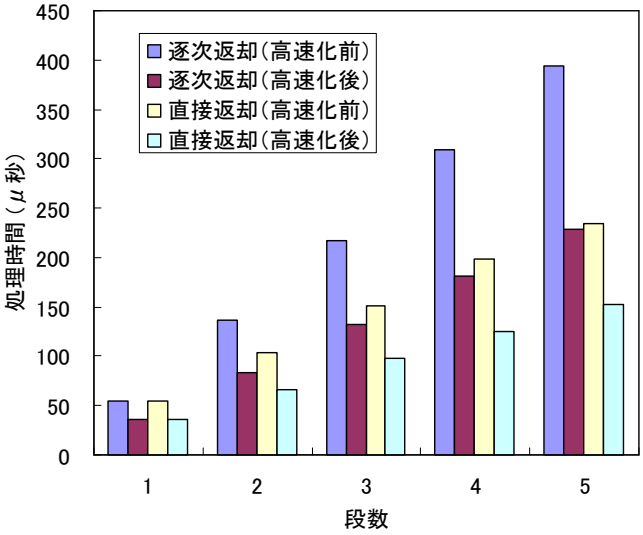


図 9 直接返却と逐次返却の比較

1MB とし，その他のページサイズを 4KB としている．これにより，ICA の利用する TLB のエントリ数を減らし，TLB ミスの発生を抑制している．

5.2 TLB preloading

文献 [8] は，アドレス変換情報を事前に TLB へ登録することで，システムの性能向上を実現している．具体的には，TLB の利用履歴を管理し，TLB ミスが発生すると予測されるアドレス変換情報を事前に TLB へ登録する．これにより，TLB ミスの発生を抑制している．これに対し，提案した高速化手法では，AP プロセスの利用する ICA のアドレス変換情報を事前に TLB へ登録している．これにより，ICA へのアクセスにおいて，TLB ミスは発生しない．

5.3 L4-Linux

L4-Linux は，マイクロカーネル構造を持つ OS である．L4-Linux は，多重仮想記憶を採用しており，Linux のカーネルや AP は，独立した仮想空間上で，プロセスとして動作する．ここで，L4-Linux において，L4 マイクロカーネルは，プログラム間通信の制御を担当し，Linux カーネルに相当するプロセス（以降，Linux カーネルプロセス）は，メモリ領域の管理，ファイルの管理，および割込の管理を担当する．このため，Linux カーネルプロセスと AP プロセス間で通信が頻繁に発生する．この性能低下を防ぐために，L4-Linux では，

(1) マッピングを利用したバッファの受渡し
(2) 少ない引数のレジスタ渡し
(3) キャッシュミスの低減

などの対処 [9][10] を行っており，性能を向上させている．これに対し，提案した高速化手法では，ICA のアドレス変換情報を TLB に常駐させることにより，通信中に発生する TLB ミスを抑制している．これにより，TLB ミスの発

生に伴う処理オーバヘッドを削減し，サーバプログラム間通信の性能を向上させている．

6. まとめ

AnT を単に SH-4 へ移植しただけでは，***AnT*** のサーバプログラム間通信機構は Pentium4 用に設計されており，SH-4 の特徴は生かされていないため，高性能を期待できない．そこで，SH-4 向けサーバプログラム間通信機構の高速化手法を提案し，高速化の効果を明らかにした．

提案した手法は，ICA のアドレス変換情報を TLB に常駐させる．また，OS サーバは全 ICA を常に読み書き可能とし，AP プロセスが利用する ICA サイズを制限することにより，ICA の管理を簡素化する．さらに，TLB エントリを分割管理し，エントリ指定での TLB へのアドレス変換情報の登録を行う．評価により，サーバプログラム間通信機構の処理依頼時間で平均約 50%，結果返却時間で平均約 42%短縮できることを示した．また，OS サーバを n 段介した処理依頼において，段数に関係なく，逐次返却と直接返却の時間を約 40%短縮できることを確認した．

残された課題として，高速化手法をサービス事例で評価することがある．

参考文献

[1] D.L. Black, D.B. Golub, D.P. Julin, R.F. Rashid, R.P. Draves, R.W. Dean, A. Forin, J. Barrera, H. Tokuda, G. Malan, and D. Bohman, “Microkernel Operating System Architecture and Mach,” J. Inf. Process., vol.14, no.4, pp.442-453, 1992.

[2] J. Liedtke, “Toward Real Microkernels,” Commnun. ACM, vol.39, no.9, pp.70-77, 1996.

[3] A.S. Tanenbaum, J.N. Herder, and H. Bos,“Can we make operating systems reliable and secure?,”IEEE Computer Magazine, vol.39, no.5, pp.44-51, 2006.

[4] 岡本幸大，谷口秀夫，“***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価,”電子情報通信学会論文誌 (D)，Vol.J93-D，No.10，pp.1977-1989，2010.

[5] 鶴谷昌弘，山内利宏，谷口秀夫，“***AnT*** オペレーティングシステムの SH-4 への移植,”情報処理学会研究報告，Vol.2011-OS-117，No.3，pp.1-8，2011.

[6] 横山和俊，乃村能成，谷口秀夫，丸山勝巳，“応用プログラムの走行モード変更を可能にするプロセス制御機構,”電子情報通信学会論文誌 (D)，Vol.J91-D，No.3，pp.696-708，2008.

[7] T.H. Romer, W.H. Ohlrich, A.R. Karlin, B.N. Bershad, “Reducing TLB and memory overhead using online superpage promotion,”Proc. ISCA-22, vol31. no.9, pp.176-187, 1995.

[8] A. Saulsbury, F. Dahlgren, P. Stenstrom, “Recency-based TLB preloading,” Proc. ISCA-27, vol.28, no.2, pp.117-127, 2000.

[9] J. Liedtke, “Improving IPC by kernel design,” Proc. SOSP-14, vol.27, no.5, pp.1-15, 1993.

[10] H. Hartig et al.,“The performance of μ-based systems,” Proc. SOSP-16, vol.31, no.5, pp.1-15, 1997.