# OSにおける割込み処理のオーバヘッドを軽減する一括割込み処理機構

乃村能成[†] 中島耕太[††,☆]
谷口秀夫[†††] 雨宮真人[†]

近年，計算機の高性能化が著しく，それにともない計算機周辺の入出力装置の処理速度も向上してきている．また，計算機の取り扱うデータ量の増加により，入出力の頻度も増加してきている．入出力の頻度の増加は，割込み発生の回数の増加をともなう．このため，割込み処理のオーバヘッドを削減し，効率の高い割込み処理を実行する必要がある．そこで，我々は，割込み処理のオーバヘッドを削減する手法として，複数の割込みを一括処理する一括割込み処理機構を提案する．我々は，一括割込み処理機構を Myrinet（1.28 Gbps）通信用の NIC（Network Interface Card）からの割込み処理に対して設計，実装を行った．本論文では，一括割込み処理機構の設計について述べ，実現した一括割込み処理機構の基本性能，および，他の演算処理を並列に実行させた場合の相互影響について報告する．

# Interrupt Packaging Mechanism to Reduce Overheads of Interrupt

Yoshinari Nomura,[†] Kohta Nakashima,[††,☆] Hideo Taniguchi[†††]
and Makoto Amamiya[†]

According to the increasing amount of data transferred between computers and peripheral devices, the frequency of interrupts from peripheral devices increases. Thus, efficiency of interrupt handling is one of the key issues to realize high performance computing environments. In order to reduce interrupt overheads, we propose an interrupt packaging mechanism that aggregates main handlers of a series of interrupts. We have designed and implemented an interrupt packaging mechanism for interrupts from NIC (Network Interface Card) of Myrinet (1.28 Gbps). In this paper, we report design and implementation of an interrupt packaging mechanism, and our performance evaluation.

## 1. はじめに

近年，計算機の高速化が著しく，それにともない計算機周辺の入出力装置の処理速度も向上してきている．また，計算機の取り扱うデータ量の増加により，入出力の頻度も増加してきている．入出力の頻度の増加は，割込み発生回数の増加をともなう．割込み発生回数が増加すると，他の演算処理の性能低下といった影響を与える恐れがある．このため，割込み処理のオーバヘッドを削減し，効率の高い割込み処理を実行する必要がある．

一般に，CPUは，入出力装置への入出力要求後，入出力装置からの結果が得られるまでの間，他の演算処理を実行し，入出力装置からの結果が得られたことを割込みにより検知し，入出力処理を実行している．割込み処理は，実行中の他の演算処理に割り込んで実行されるため，レジスタの退避や回復といった，前処理や後処理を必ず実行する必要がある．

そこで，我々は，割込み処理の前処理と後処理のオーバヘッドを削減する手法として，複数の割込みを一括処理する一括割込み処理機構を提案する．この機構では，割込み発生ごとに割込み処理を行わず，割込み発生時には，割込み発生の登録処理のみを行い，後に一括して登録情報をもとに割込み処理を行う．このように一括して割込みを処理することにより，従来は各割

† 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University
†† 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
††† 岡山大学工学部
Faculty of Engineering, Okayama University
☆ 現在，株式会社富士通研究所
Presently with Fujitsu Laboratories, LTD.

込み発生ごとに実行した前処理と後処理の削減をはかる．また，複数の同種類の割込みが発生した場合には，後に一括して割込み処理を行うことにより，同じ割込み処理ルーチンを連続して実行することになる．この場合，割込み処理ルーチンのキャッシュのヒット率が向上し，システム全体の処理性能のさらなる向上が期待できる．

本論文では，一括割込み処理機構の設計について述べ，実現した一括割込み処理機構の性能評価について報告する．

ここでは，本機構を ***Tender*** オペレーティングシステム[1] の割込み制御部に実装して評価した．

なお，この一括割込み処理方式は，OSが割込みを扱う処理方式に関する提案であり，OSが持つ機能に依存するものではない．そのため，大半のOSの割込み処理に適用可能である．我々は，本機構を並列分散オペレーティングシステム CEFOS[2] にも同時に実装中であり，細粒度マルチスレッディング処理への適用性の評価を進めている．

## 2. 逐次割込み処理と一括割込み処理

### 2.1 概　　要

従来の割込み処理では，割込みが発生すると，この割込みに対応する割込み処理を行う．この処理は，通常以下のように行われる．まず，割込みが発生すると，割込み処理ルーチンは，レジスタ群の退避や，割込みコントローラへの設定処理といった前処理を行う．次に，各割込みに対応する処理である本処理を行う．最後に，割込みコントローラへの処理やレジスタの回復処理といった後処理を行う．このように，割込みが発生すると，必ず本処理の前後に前処理と後処理を実行する．この処理方式を，以降，逐次割込み処理方式と呼ぶ．

逐次割込み処理方式では，割込み発生ごとに必ず前処理と後処理を実行する．このため，割込みが頻発するような環境では，前処理や後処理の負荷がシステムの負荷全体に占める割合が増加する．そこで，割込み発生ごとに必要となる前処理，後処理の処理時間を削減する一括割込み処理方式を提案する．

一括割込み処理方式の概要を述べる．割込み発生時には，割込みが発生したことを記録する割込み登録処理のみを行う．この登録処理は，割込みの発生を記録するのみであるため，レジスタの退避/回復処理や割込みコントローラへの設定処理の一部を省略できる．このように登録された割込みを後で一括して処理する．複数の割込みを一括して処理することにより，逐次割込み処理の際の，前処理，後処理の処理時間を削減する．また，同種類の割込みが複数発生した場合には，一括処理実行時に同じ本処理を複数回実行することになる．このような場合においては，キャッシュのヒット率が向上し，性能改善が見込める．

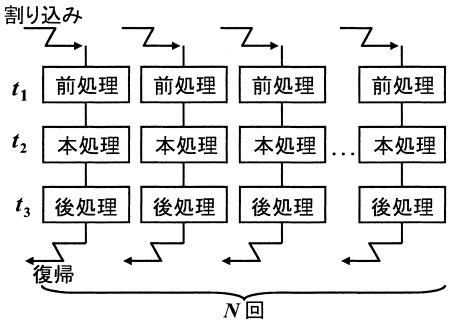


**図 1** 逐次割込み処理方式での命令実行
Fig. 1 Sequential interrupt handling.

一括処理の契機は，以下の条件のいずれかが満たされた場合である．

(1) 登録された割込みが $N$ 回だけ溜まったとき
(2) ある割込みが登録されてから未処理のまま時間 $T$ が経過したとき
(3) 逐次処理が必要な割込みが発生したとき

$N$ や $T$ はデバイスの特性によって設定する．

もちろん，これらの条件を組み合わせることが必要な場合もある．なお，$N$ や $T$ はデバイスの特性によって変更可能である．たとえば，割込みが頻発するデバイスでは，多数の割込み処理を待たせないために，$N$ を重視した契機とする必要がある．つまり，一括処理可能な，割込み数の最大値 $N$ を満足した時点で，一括処理を開始する．一方，実時間処理を必要とするデバイスでは，割込み発生から，割込み処理開始までの時間が制約されるため，$T$ を重視した契機とする必要がある．つまり，割込みが発生してから，一括処理を行うまでの時間を $T$ 以下にする必要がある．

### 2.2 総実行時間の比較

逐次割込み処理方式による場合と，一括割込み処理方式による場合との割込み処理の総実行時間の比較について述べる．逐次割込み処理方式による場合の命令実行の様子を**図 1**，一括割込み処理方式による場合の命令実行の様子を**図 2** に示す．

逐次割込み処理方式では，前処理の処理時間を $t_1$，本処理の処理時間を $t_2$，後処理の処理時間を $t_3$ とすると，$N$ 回割込みが発生した場合の割込み処理の総実行時間は，

$$N(t_1 + t_2 + t_3) \tag{1}$$

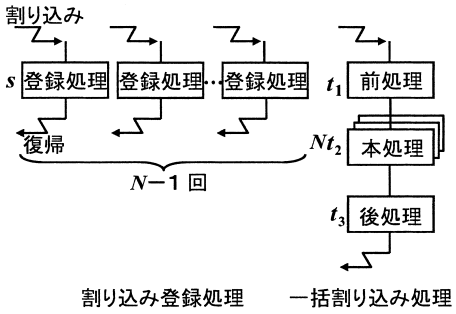


**図 2** 一括割込み処理方式での命令実行
Fig. 2 Packed interrupt handling.

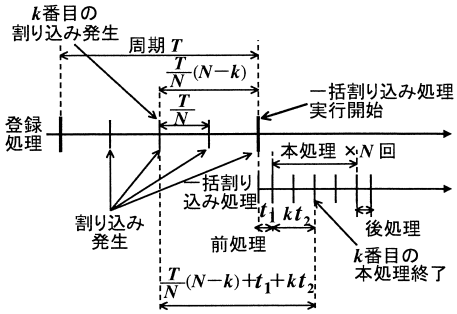


**図 3** 一括割込み処理処理方式の処理の時間経過
Fig. 3 Timing chart for packed interrupt.

である．

一方，一括割込み処理方式では，$N$ 回の割込みを契機に一括処理を行うとすると，登録処理は $N-1$ 回行われる．そして，$N$ 回目の割込みが発生した際に，前処理を 1 回，本処理を $N$ 回，後処理を 1 回実行する．登録処理の処理時間を $s$，前処理の処理時間を $t_1$，本処理の処理時間を $t_2$，後処理の処理時間を $t_3$ とすると，$N$ 回割込みが発生した場合の割込み処理の総実行時間は，

$$(N-1)s + t_1 + Nt_2 + t_3 \tag{2}$$

である．

逐次割込み処理方式と，一括割込み処理方式の総実行時間の差は，式 (1)− 式 (2) より，

$$(N-1)(t_1 + t_3 - s) \tag{3}$$

である．ここで，

$$t_1 + t_3 - s > 0 \tag{4}$$

であるならば，式 (3) は正である．つまり，式 (4) の条件を満たす場合，すなわち，前処理と後処理の処理時間の和より登録処理の処理時間が短い場合，割込み処理を一括することにより処理時間を短縮できる．

### 2.3 割込み平均遅延時間の比較

逐次割込み処理方式による場合と，一括割込み処理方式による場合との割込みの平均遅延について述べる．

逐次割込み処理方式の場合，割込み発生直後に前処理，本処理，後処理を実行するため，割込みが発生してから各割込みの本処理が終了するまでの時間は，

$$t_1 + t_2 \tag{5}$$

である．ここで，OS の割込み処理では，処理している割込みと同じ割込みが入った場合への対処として，大きく 2 つある．1 つは，割り込まれたレベルで割込み処理のプログラムを実行するもので，同じ割込みはブロックされる．もう 1 つは，割り込まれたレベルでの割込み処理のプログラム実行を最小限とし，割込み処理の半ばでプログラム走行レベルを低設定し，同じ割込みの発生を許すものである．一括処理は，上記の 2 つの方法のいずれでも実現可能であり，方法選択は，従来の割込み処理の方法を選択する際と同じ観点になる．なお，以下の記述では，一括処理は割り込まれたレベルで行われるものと仮定した．

一方，一括割込み処理方式の場合，割込み処理実行の処理の時間経過を**図 3** に示す．等間隔で時間 $T$ の間に $N$ 回割込みが発生し，また，$N$ 回目の割込み発生時に一括割込み処理を実行すると仮定する．なお，図 3 では，$N=4$ の場合を示している．このように仮定すると，割込み発生の間隔は，

$$\frac{T}{N} \tag{6}$$

となる．この期間に発生した割込みはこの期間が終了するときに一括して処理される．この期間の第 $k$ 番目に発生した割込みに対する処理は，一括処理の開始まで待たされる．図 3 では，$k=2$ の場合を示している．第 $k$ 番目に発生した割込みが発生してから，一括処理が起動されるまでの時間は，式 (6) より

$$\frac{T}{N}(N-k) \tag{7}$$

である．さらに，一括処理の開始から第 $k$ 番目に発生した割込みに対応する処理の実行が完了するまでにかかる時間は，

$$t_1 + kt_2 \tag{8}$$

である．すなわち，第 $k$ 番目の割込み発生から本処理が終了するまでの時間は，式 (7) と式 (8) より，

$$\frac{T}{N}(N-k) + t_1 + kt_2 \tag{9}$$

である．したがって，この期間に発生した割込みからそれぞれに対応する割込み処理が実行されるまでの平均時間は，式 (9) より，

$$\frac{1}{N}\sum_{k=1}^{N}\left\{\frac{T}{N}(N-k) + t_1 + kt_2\right\} \tag{10}$$

**表 1** 逐次割込み処理方式と一括割込み処理方式の比較
Table 1 Comparison between sequential interrupt and packed interrupt.

| 割込み処理方式 | 長所 | 短所 |
|---|---|---|
| 逐次割込み処理方式 | 割込み発生から割込み処理実行までの時間が短い | 割込み処理の総実行時間が長い |
| 一括割込み処理方式 | 割込み処理の総実行時間が短かい | 割込み発生から割込み処理実行までの時間が長い |

である．これを整理すると，

$$\frac{1}{2}\left(\frac{N-1}{N}\right)T + t_1 + \frac{1}{2}(N+1)t_2 \qquad (11)$$

となる．

逐次割込み処理方式と一括割込み処理方式を比較すると，一括割込み処理方式と逐次割込み処理方式の平均的な遅延の差は，式 (11)− 式 (5) より，

$$\frac{1}{2}\left(\frac{N-1}{N}\right)T + \frac{1}{2}(N-1)t_2 \qquad (12)$$

となる．

### 2.4 総合比較

逐次割込み処理方式と一括割込み処理方式の長短を**表 1** に示す．逐次割込み処理方式は，割込みが発生してから即座に本処理を行うので本処理実行終了までの時間が短い．一方，すべての割込みに対して，全汎用レジスタの退避や回復，割込みコントローラへの設定処理といった前処理や後処理を行う必要があり，総実行時間は，一括割込み処理方式より長い．

一括割込み処理方式は，割込み登録処理時に，逐次割込み処理方式での前処理や後処理のオーバヘッドが削減できる．つまり，一般に式 (4) が成り立つ．このため，総実行時間は，逐次割込み処理方式より短かい．一方，割込みが発生してから一括処理を実行するまで遅延があるため，割込みが発生してから本処理実行までの時間が逐次割込み処理方式と比べて，長い．遅延の差の平均は，式 (12) で表され，$N > 1$ においてこの式が正になることからも分かる．

このように，逐次割込み処理方式は，応答時間重視である．実時間処理を行う場合等，応答時間が重要視される場合には，逐次割込み処理方式の方が適している．

一方，一括割込み処理方式は，スループット重視である．入出力回数が頻発するが，応答時間はあまり重要ではない場合には，一括割込み処理方式の方が適している．例として，プロセススケジューラがタイムスライスのみをサポートし，プリエンプションをサポートしないシステムの場合を考察する．この様子を**図 4** に示す．図 4 では，タイムスライス間隔が 1.0 秒で，優先度 5 のプロセス A は，1.0 秒プロセッサ処理を行い，入出力待ち 0.5 秒を繰り返すとする．優先度 3 のプロセス B は，プロセッサ処理を実行し続けるものとする．

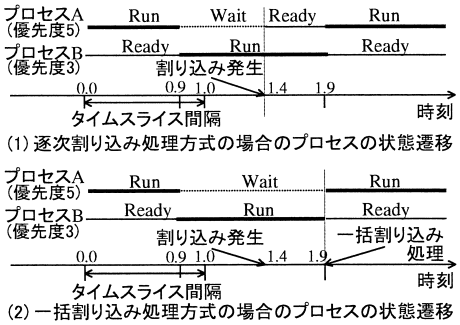


**図 4** プロセス状態の遷移
Fig. 4 Statechart for each interrupt handling.

このシステムで，逐次割込み処理方式で割込み処理を行った場合について述べる．まず，時刻 0.0 秒からプロセス A が Run 状態となり，プロセス B は Ready 状態となる．時刻 0.9 秒の時点で，プロセス A は入出力待ちとなり，Wait 状態に遷移する．また，これにともない，Ready 状態のプロセス B が Run 状態に遷移する．時刻 1.4 秒の時点で，入出力完了割込みが発生し，割込み処理によりプロセス A は Wait 状態から，Ready 状態に遷移する．時刻 1.9 秒の時点で，タイムスライス処理が実行され，プロセス A が Run 状態へ，プロセス B が Ready 状態へ遷移する．

次に，一括割込み処理方式で割込み処理を行った場合について述べる．ここでは，一括処理の契機をタイムスライス処理の実行直前とする．まず，時刻 0.0 秒からプロセス A が Run 状態となり，プロセス B は Ready 状態となる．時刻 0.9 秒の時点で，プロセス A は入出力待ちとなり，Wait 状態に遷移する．これにともない，Ready 状態のプロセス B が Run 状態に遷移する．時刻 1.4 秒の時点で，入出力完了割込みが発生するが，ここでは割込み登録処理のみが行われる．したがって，プロセスの状態は遷移しない．時刻 1.9 秒の時点で，一括割込み処理が実行される．このため，プロセス A は Ready 状態に遷移する．さらに，タイムスライス処理が実行され，プロセス A が Run 状態へ，プロセス B が Ready 状態へ遷移する．この場合，割込み処理が遅延することによって，プロセス A の走行が遅延することはなく，逐次割込み処理の場合と同様にプロセス A が走行する．このように逐次割込

み処理方式の場合とタイムスライスを契機として一括処理を行った場合とでは，入出力待ちのプロセスにプロセッサが割り当てられるまでの時間は変わらない．したがって，一括割込み処理方式の短所である割込み発生から割込み処理実行までの遅延時間が長いという問題が隠蔽され，この短所がシステムの応答時間に悪影響を与えていない．このような場合では，一括割込み処理方式は大きく効果を発揮する．

UNIX では，タイムスライス機能とプリエンプション機能があるが，プリエンプション機能については，カーネル内を走行中はプリエンプション不可である．カーネル内には，TCP/IP といった通信プロトコル処理が実装されており，ソケットシステムコール内では非プリエンプション時間が長い．したがって，通信が頻発する場合には，プリエンプションされる場合が減少する．その結果として一括割込みによるスケジューリングへの影響を小さく抑えることができるため，上記のように一括割込み処理方式が大きく効果を発揮する可能性がある．

また，異なる割込みが何らかの順序関係や依存関係を持っている場合，これらを考慮した実装を行う必要がある．たとえば，端末へのキー入力割込みとそのエコーバックのための表示系の割込みの取扱いでは，依存関係を考慮しなければならない．本論文では，これらについては考慮の対象となっていない．

## 3. 一括割込み処理機構

### 3.1 一括割込み処理機構の構成

一括割込み処理機構の構成を**図 5** に示す．一括割込み処理機構は，各割込み発生ごとに割込みを登録する割込み登録処理と，一括して各割込みに対応する処理を行う一括処理からなる．割込み登録処理では，発生した割込みに関する情報を割込み登録情報表と揮発性データ格納域に記録し，一括処理時にこれらを参照する．

以降，割込み登録処理と一括処理について述べる．

### 3.2 割込み登録処理

まず，逐次割込み処理方式での割込み発生時の処理について述べる．ここでは，例として PC/AT 互換機での場合について述べる．割込み処理は以下のように実行される．

(1) 全汎用レジスタの退避処理（前処理）
(2) 割込みコントローラへの割込み受け取り通知処理（前処理）
(3) 割込みマスク変更処理（前処理）
(4) 割込み禁止解除処理（前処理）

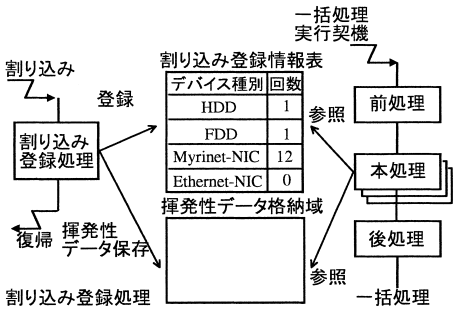


**図 5** 一括割込み処理概要
Fig. 5 Abstract of packed interrupt mechanism.

(5) 本処理
(6) 割込み禁止処理（後処理）
(7) 割込みマスク復元処理（後処理）
(8) 全汎用レジスタの回復処理（後処理）

割込みが発生し，登録されている割込み処理ルーチンが起動されるときには，割込み禁止状態である．割込み禁止状態の間に，処理 (1)，(2) および (3) を行う．全汎用レジスタの退避処理により，以降の処理で汎用レジスタを制限なく使用できる．割込みコントローラへの割込み受け取り通知処理により，割込みコントローラは，割込みを再び受付けることが可能になる．割込みマスク変更処理により，本処理中で許可/不許可する割込みを指定することが可能になる．処理 (6)，(7) および (8) は，前処理に対する後処理であり，割込みマスクや汎用レジスタの状態を割込み発生前の状態に復元するための処理である．

次に，一括割込み処理方式での割込み登録処理について述べる．割込み登録処理で必要となる処理は，以下のとおりである．

(1) 割込み登録処理で使用するレジスタの退避処理
(2) 割込み種別の確認
(3) 割込みコントローラへの割込み受け取り通知処理
(4) 割込み発生の記録処理
(5) ハードウェア上の揮発性データの保存処理
(6) 割込み登録処理で使用するレジスタの回復処理

まず，(1)，(6) の割込み登録処理で使用するレジスタの退避・回復処理は，逐次割込み処理の (1)，(8) に対応する．逐次割込み処理においては，全汎用レジスタに関しての処理が必要があるのに対し，一括割込み処理においては，逐次割込み処理における (5) の本処理を行わないため，割込み登録処理で使用するレジスタのみの退避・回復処理で済む．これにより，レジスタ退避・回復処理の軽減をはかる．(2) の割込み種別の確認は一括割込みのために新たに必要な処理で

ある．2.4 節で述べたように，割込み種別により，一括割込み処理方式を適用すべきかどうかを判断する部分である．割込み回線を複数種類の割込み種別で共有している場合には，割込み登録処理において，割込み種別を判別し，逐次割込み処理方式によって処理すべき割込みに対しては，その時点で，逐次割込み処理方式による割込み処理を実行する．このようにすることにより，各割込み種別に対して，適切な方式を適用する．逐次方式の (3)〜(7) に対して，一括方式では，(4)，(5) を行っている．逐次処理の (3)，(7) で必要であった割込みマスクの変更/復元処理について，一括方式では，割込み登録処理中は他の割込みの受付けは不要なため，前処理と後処理での割込みマスクの変更/復元処理を省略している．また，(4) の割込み発生の記録処理では，割込みが発生したことを記録する処理を行う．この処理では，各デバイスごとの割込み発生回数を割込み登録情報表に記録する．(5) のハードウェア上の揮発性データの保存処理では，ハードウェア上のデータのうち，次の割込みが発生すると消失する揮発性データを保存するため，ハードウェア上の揮発性データを揮発性データ格納域に保存する．

なお，新規処理の 1 つである，処理 (5) において，不揮発データがどの程度発生するかについては，デバイスや処理の内容に依存する．本一括割込み処理方式は，大量の不揮発データを処理対象としてカーネルが計算等をすることは行わず，そのままのデータをコピーすることにとどめ，より上位の層で処理することを意図している．ビデオ画像処理のようなリアルタイム処理が要求される場面においては，フレームバッファを直接アプリケーションが利用してバッファのコピーを最小限に抑えることもあるが，本一括割込み方式では，このような用途は意図していない．

### 3.3 一括処理

また，本処理は，割込み発生順序に従っては実行せず，割込み種別ごとに実行する．たとえば，Myrinet-NIC，HDD，Myrinet-NIC の順で割込みが発生した場合は，Myrinet-NIC に対する本処理を 2 回連続実行し，後で HDD に対する本処理を行う．したがって，同一種別の割込みに対する本処理は，連続して実行される．ただし，同一種別の割込みについては，実行順序は，割込み発生順序に従う．このように実行することで，本処理のキャッシュヒット率を向上させる．

## 4. 実装と評価

### 4.1 実装内容

我々の研究室で開発を行っている ***Tender*** オペレーティングシステムの割込み制御部に一括割込み処理機構を実装した．一括割込み処理機構の実装対象デバイスを Myrinet[3] 通信を制御するハードウェア（以降，Myrinet-NIC と呼ぶ）とした．Myrinet-NIC は，受信完了時に割込みを発生させる．したがって，割込み処理では受信完了処理を行う．

割込み登録処理では，割込みの発生回数を記録する処理と，次の通信が発生すると前の通信時のデータが消失する揮発性のデータの保存処理を行う．Myrinet-NIC の場合，次の通信が発生すると消失する揮発性のデータは，Myrinet-NIC 上の記憶領域にある．Pentium II 450 MHz の実測によると，Myrinet-NIC 上の領域から主記憶への 4 B のデータ複写時間は 0.540 $\mu$s であった．Myrinet-NIC 上にある次の通信が発生すると消失する揮発性のデータは，合計 8 B である．

一括処理は，タイマ割込みを契機に実行する．本実装では，タイマの周期は 10 ms とした．一括割込み処理では，割込み登録処理により登録されたデータを基に，本処理を一括して実行する．本処理では，受信完了処理を実行する．受信完了処理としては，受信サイズの取得処理，受信プロセスを Wait 状態から Ready 状態に変更する処理がある．この際，次の通信が発生すると消失する揮発性のデータは，登録処理時に主記憶に保存されているので，本処理が短縮され，逐次割込み処理を行う際よりも，処理が高速化する．

### 4.2 測定環境

***Tender*** に実現した一括割込み処理機構の評価と，比較のために逐次割込み処理機構の評価を行った．評価は，各処理部の実行時間の測定と，入出力処理時に他の演算処理を並列させて走行させた場合の相互影響の測定を行った．

測定には，Pentium II 450 MHz，主記憶 128 MB の PC/AT 互換機 2 台を Myrinet で接続した環境を用いた．Myrinet-NIC は，PCI バス（32 bit，33 MHz）で計算機と接続されている．

### 4.3 基本性能

基本性能として，一括割込み処理機構の登録処理と一括割込み処理の前処理，本処理，後処理の処理時間を測定した．本処理の測定は，1 回分の割込みに対する処理を測定した．また，比較のため，逐次割込み処理の前処理，本処理，後処理の処理時間を測定した．測定には，CPU のハードウェアクロックカウンタを用い，2 カ所のカウンタ値の差分をクロック数で割ることにより，処理時間を算出した．この測定は，50 回行い，1 回あたりの平均を算出した．結果を**表 2** に示す．

**表 3** 割込み処理平均遅延時間の比較
Table 3 Average delay of interrupt handling.

| 場合 | 一括割込み処理 ($\mu$s) | 逐次割込み処理 ($\mu$s) |
|---|---|---|
| 一括処理周期 $T\mu s$,<br>一括処理数 $N$ 回の場合 | $\frac{1}{2}\left(\frac{N-1}{N}\right)T$+8.09$N$+11.09 | 19.77 |
| 一括処理周期 10 ms,<br>一括処理数 10 回の場合 | 4,592 | 19.77 |

**表 2** 基本性能と割込み処理総実行時間
Table 2 Basic performance of interrupt handling.

| モジュール | 一括割込み<br>処理方式 ($\mu$s) | 逐次割込み<br>処理方式 ($\mu$s) |
|---|---|---|
| 割込み登録処理 | 2.80 | — |
| 前処理 ($t_1$) | 3.00 | 3.00 |
| 本処理 ($t_2$) | 16.17 | 16.77 |
| 後処理 ($t_3$) | 1.82 | 1.82 |
| $N$ 回の処理時間の合計 | $18.97N + 2.02$ | $21.59N$ |
| 10 回の処理時間の合計 | 191.7 | 215.9 |

4.1 節で述べたように，逐次割込み処理の本処理と，一括割込み処理の本処理とを比較すると，登録処理時に次の通信が発生すると消失する揮発性データを主記憶に保存しているので，一括割込み処理の本処理の方が高速である．また，逐次割込み処理の前処理時間と後処理時間の和は，4.82 $\mu$s であり，一括割込み処理の割込み登録処理時間は，2.80 $\mu$s であるため，式 (4) を満たす．すなわち，一括割込み処理方式の方が逐次割込み処理方式よりも，割込み処理の総実行時間が短くなることを示している．

### 4.4 割込み処理総実行時間の比較

割込み処理総実行時間の比較を表 2 に示す．$N$ 回の処理時間の合計と 10 回の処理時間の合計については，表 2 で示した数値を基に，式 (1) と式 (2) を用いて算出した式または数値である．

$N$ 回の処理時間の合計について一括割込み処理方式と逐次割込み処理方式を比較すると，$N$ の係数の値が，一括割込み処理方式の方が小さい．これは，一括する割込みの個数が増加すると，逐次割込み処理方式と比較した場合の一括割込み処理方式の総処理時間の削減分が増加することを意味する．両者が同じ時間になる $N$ は，$N = 1.25$ である．つまり，$N \geq 2$ のとき，一括割込み処理時間が短くなる．具体的には，10 回の割込みを一括して処理すると，約 11.2%に相当する 24.2 $\mu$s 処理時間が削減できることが期待できる．

### 4.5 割込み処理平均遅延時間の比較

割込み処理平均遅延時間の比較を**表 3** に示す．表 3 において，一括処理周期 $T\mu$s，一括割込み処理数 $N$ 回の場合は，一括割込み処理方式については，一括処理周期 $T\mu$s，一括割込み処理数 $N$ 回の場合の割込み処理平均遅延時間を示し，逐次割込み処理方式については，割込み処理の平均遅延時間を示している．一括処理周期 10 ms，一括割込み処理数 10 回の場合は，一括割込み処理方式については，一括処理周期 10 ms，一括割込み処理数 10 回の場合の割込み処理平均遅延時間を示し，逐次割込み処理方式については，割込み処理の平均遅延時間を示している．これらの割込み処理平均遅延時間は，式 (5) と式 (11) を用いて算出した式または数値である．

一括処理周期 $T$ や，一括割込み処理数 $N$ が増加すると，一括割込み処理時の割込み処理平均遅延時間は増加する．具体的には，一括処理周期が 10 ms であり，一括割込み処理数が 10 回の場合には，割込み処理平均遅延時間は，4,592 $\mu$s となり，逐次割込み処理の遅延時間（19.77 $\mu$s）と比較すると非常に大きい．このため，一括割込み処理方式では，実時間処理といった応答遅延時間に厳しい制約を求められるシステムには不向きである．

一方，2.4 節で述べたように，UNIX では，カーネル内走行中はプリエンプション不可であり，ソケットシステムコール内では非プリエンプション時間が長い．さらに，プリエンプション機能をサポートしないような，応答遅延時間の制約が比較的緩いシステムでは，あまり大きくシステム全体の性能に影響を与えないものと考えられる．たとえば，プリエンプション機能をサポートしないシステムでタイムスライス間隔が 10 ms である場合，逐次割込み処理方式では，割込み発生から入出力待ちのプロセスに CPU が割り当てられるまでの時間は，平均 5,000 $\mu$s である．この場合と一括割込み処理方式の平均遅延時間（4,592 $\mu$s）を比較すると，大きな遅延ではない．逆のいい方をすると，CPU が割り当てられるまでの平均時間と一括割込み処理方式の平均遅延時間に大差が生じないように一括割込み処理数を決定する必要がある．

### 4.6 他の演算処理との相互影響評価

逐次割込み処理方式を用いた場合と一括割込み処理方式を用いた場合における，割込み処理が他の演算処理に与える影響について評価を行った．

以下のように評価を行った．割込み発生をともなう

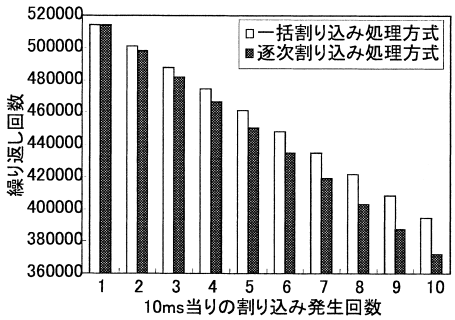


**図 6** 演算処理の繰返しの回数
Fig. 6 Influence of each interrupt on other processes.

入出力処理として Myrinet によるデータ受信を繰り返す処理を実行する．また，割込み発生をともなわない演算処理として，ループ処理を繰り返し，その繰返しの回数をカウントする処理を一定時間実行する．これらの入出力処理と演算処理を並列に実行させ，演算処理量として演算処理が一定時間にカウントする繰返しの回数を測定した．すなわち，他の演算処理の演算処理量が多いということは，入出力処理が演算処理に与える影響が小さく，性能が高いということを示している．なお，入出力処理は，演算処理より高いプロセス優先度で実行した．この測定を，逐次割込み処理方式を用いた場合と，一括割込み処理方式を用いた場合についてそれぞれ行った．

測定では，演算処理の実行時間を 1 s とし，一括処理を 10 ms おき実行することにした．また，一括処理周期である 10 ms 間に発生する割込みの回数を 1 回から 10 回まで変化させた各場合について測定を行った．結果を**図 6** に示す．

10 ms 間に発生する割込みの回数を増加させると，各場合とも，演算処理量は減少する．これは，入出力処理の負荷が増加するため，演算処理に対して与える影響が増加するためであると考えられる．

また，一括処理する割込みの回数が 2 回以上の場合は，一括割込み処理方式の方が，逐次割込み処理方式よりも演算処理量が多い．

そして，一括処理する割込みの回数が 10 回の場合において，一括割込み処理方式では，約 394,000 回ループを実行できているのに対して，逐次割込み処理方式では，約 372,000 回である．したがって，一括割込み処理方式の場合は，システム全体の性能は，約 5.91%改善される．一方，4.4 節で示した結果を基に計算すると，1 s 間に一括割込み処理が 100 回実行されるので，実行時間は，2.42 ms 削減できると期待できる．なお，1 s 間に入出力処理をともなわない場合，演算処理が実行するループ処理の繰返し回数は，約 532,000 回であった．これらから，一括割込み処理方式を用いた場合は，約 1,290 回（$= 532000 \times 2.42/1000$）多くループ処理を実行できると期待できる．すなわち，予測値によると，性能改善率は，約 0.3%である．これと比較すると先に示した実際の測定による性能改善率（約 5.91%）は，非常に高い．

これにより，一括割込み処理方式では，同種の割込みが複数回発生した場合，同じ本処理の命令群を連続して実行するためキャッシュのヒット率が向上して入出力処理の処理負荷が軽減され，その結果，性能改善率が予測を上回ることを確認できた．

## 5. 関 連 研 究

OS は，古くから割込み処理の効率化のためのさまざまな工夫を行ってきた．しかし，割込み処理をまとめて行う機構を持つ OS は，マイクロプロセッサを利用した単一プロセッサの計算機では数少ない．現在の代表的な OS として，Linux がある．もともと，いくつかの UNIX 系 OS では，割込みを処理を分割し，カーネルモードからユーザモードへ移行する際に処理をまとめて行う方式が採用されている．具体的な実装は個々に違うものの，「カーネルモードからユーザモードへ移行する際に処理をまとめて行う」点は，共通である．

たとえば，Linux には，tasklet（あるいは，ボトムハーフ）と呼ばれる機構が存在する．tasklet とは，デバイスドライバがその中で宣言し，カーネルに登録する特殊な関数で，カーネルが決定した安全な時期に実行される．具体的には，ksoftirqd と呼ばれるカーネル内スレッド（プロセス）がスケジュールされた際に実行される．デバイスドライバでは，この機構を利用し，早急に実行する必要がない関数の実行をカーネルのスケジューリングに委ねる．これによって，割込み処理中に実行するコードを最小限に抑えるようにすることができる．この仕組みを利用することによって，割込み処理のオーバヘッドを低減することが可能である．しかし，この手法は関数の実行がカーネルモードでの処理の最後に行われるのに対し，提案した手法は割込み処理として行われる．このため，下記の点について提案した手法とは異なる．

(1) 一括処理回数の調整が難しい．
tasklet は，デバイスドライバからは，いつ実行されるか分からないので，一括処理する割込みの回数という考え方はない．したがって，一括処理する割込みの回数を微調整するといった，

デバイス固有の情報に基づく細かな制御をすることが難しい．

(2) ユーザモードとカーネルモードでの動作の違い．OSがカーネルモードで走行中に割込みが多発した場合は，taskletを利用することによって割込み処理をまとめることができる．そのため，(1)の違いがあるものの，一括割込みに近い効果が期待される．

しかしながら，OSがユーザモードで走行中に割込みが多発した場合は，最初の割込みでプロセスの再スケジュールが発生してしまうために，taskletの実行は，割込みごとに実行されることになる．したがって，期待した効果が得られない．

以上のことから，一括割込み回数の調整をデバイスごとに行うことができ，OSの走行モードに依存しない一括割込み制御が期待できる点で，本方法の特長があるといえる．

一方，割込み発生の回数を削減する手法として，ATMやGigabit Ethernetといった高速通信路を制御するNIC上で，複数の割込みを一括してCPUに対して通知する手法があり，この機能を利用することにより，割込み処理のオーバヘッドを削減する研究がなされている[4)~6)]．しかし，この方法を用いる場合には，NICによる割込み一括機能のサポートが必要である．また，この機能を制御する機構がOSに求められる．

## 6. おわりに

複数の割込みを一括処理する一括割込み処理機構を提案した．一括割込み処理方式を用いることで，割込み処理の前処理や後処理といったオーバヘッドを削減できる．一方，一括割込み処理方式を用いると，割込み発生から割込み処理が実行されるまでの遅延時間が増大する問題点がある．逐次割込み処理方式と一括割込み処理方式では，それぞれ長短があり，応答時間重視の処理においては逐次割込み処理方式，ファイル転送のようなスループット重視の処理においては一括割込み処理方式が優れている．

また，一括割込み処理機構の設計について述べた．一括割込み処理機構は，割込み登録処理と一括処理から構成され，割込み登録処理での処理内容や一括割込み処理での処理内容について述べた．

さらに，一括割込み処理方式を***Tender***のMyrinet制御に適用し，基本性能，および他の演算処理との相互影響を評価し，他の演算処理との相互影響評価では，10回の割込みを一括処理する場合には，他の演算処理の性能は約5.91%改善することを示した．

残された課題として，一括割込み処理機構のさまざまな入出力装置への適用や実アプリケーションに対する性能評価がある．

**謝辞** 本研究は，文部科学省科学研究費補助金，基盤研究（A）(2)「細粒度マルチスレッド処理原理による並列分散処理カーネルウェアの研究」(課題番号：15200002）による．

## 参考文献

1) 谷口秀夫，青木義則，後藤真孝，村上大介，田端利宏：資源の独立化機構による***Tender***オペレーティングシステム，情報処理学会論文誌，Vol.41, No.12, pp.3363–3374 (2000).
2) 日下部茂，富安洋史，村上和彰，谷口秀夫，雨宮真人：並列分散オペレーティングシステムCEFOS（Communication-Execution Fusion OS），信学技報，Vol.99, No.251, pp.25–32 (1999).
3) Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N. and Su, W.-K.: Myrinet—A Gigabit-per-Second Local-Area Network, *IEEE-Micro*, Vol.15, No.1, pp.29–36 (1995).
4) Druschel, P., Peterson, L.L. and Davie, B.S.: Experience with a High-Speed Network Adaptor: A Software Perspective, *Proc. ACM SIGCOMM Conference*, pp.2–13 (1994).
5) Kurmann, C., Müller, M., Rauch, F. and Stricker, T.M.: Speculative Defragmentation — A Technique to Improve the Communication Software Efficiency for Gigabit Ethernet, *Proc. 9th IEEE Symposium on High Performance Distributed Computing* (2000).
6) Chase, J.S., Gallatin, A.J. and Yocum, K.G.: End-System Optimizations for High-Speed TCP, *IEEE Communications Magazine*, Vol.39, No.4, pp.68–74 (2001).

(平成14年11月7日受付)
(平成15年 9 月5日採録)

**乃村 能成**（正会員）

昭和44年生．平成5年九州大学工学部電子工学科卒業．平成7年同大学大学院情報工学専攻修士課程修了．同年九州大学工学部助手．平成8年九州大学大学院システム情報科学研究科助手．オペレーティングシステムとソフトウェア開発環境，グループ支援環境に興味を持つ．博士（情報科学）．

**中島 耕太**（正会員）

昭和52年生．平成12年九州大学工学部電気情報工学科卒業．平成14年同大学大学院システム情報科学府情報工学専攻修士課程修了．同年富士通（株）入社．現在,（株）富士通研究所勤務．高速インタコネクトに関する研究・開発に従事．

**谷口 秀夫**（正会員）

昭和53年九州大学工学部電子工学科卒業．昭和55年同大学大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．昭和62年同所主任研究員．昭和63年NTTデータ通信（株）開発本部移籍．平成4年同本部主幹技師．平成5年九州大学工学部助教授．平成8年九州大学システム情報科学研究科助教授．平成15年岡山大学工学部教授．オペレーティングシステム，実時間処理，分散処理に興味を持つ．著書『オペレーティングシステム』（昭晃堂）等．電子情報通信学会，日本ソフトウェア科学会，ACM各会員．博士（工学）．

**雨宮 真人**（正会員）

昭和17年生．昭和42年九州大学工学部電子工学科卒業．昭和49年同大学大学院工学研究科修士課程修了．同年日本電信電話公社武蔵野電気通信研究所入所．以来，プログラミング言語・処理系，自然言語理解，データフロー・アーキテクチャ，並列分散処理，関数型/論理型言語，知能処理アーキテクチャ，マルチエージェントシステム等の研究に従事．現在，九州大学大学院システム情報科学研究院知能システム学部門教授．電子情報通信学会，ソフトウェア科学会，人工知能学会，IEEE，ACM，AAAI各会員，電子情報通信学会フェロー，情報処理学会フェロー．工学博士．