計算機アーキテクチャ 117－2
（1996．3．5）

機能別並列計算機におけるタスクの分割・並列実行

太田充彦† 河野通宗†
前沢敏行† 安西祐一郎†

機能別並列計算機では、機能ごとにタスクを分けて記述し、それぞれを機能別モジュール上で実行させる。しかし、この記述や操作は繁雑であり容易ではなかった。そこで本研究では、複数のスレッドを含むひとつのユーザタスクを実行時にモジュールごとに分割し、並列実行させる機構を設計する。また、特定のデバイスを頻繁にアクセスするスレッドが、そのデバイスの存在するモジュール上で実行されるようにタスクを分割する。これによって、デバイスを制御するサーバとユーザスレッドとのモジュール間通信の量を削減できると考える。

上記の機構を パーソナルロボット用 OS *μ–PULSER* 上に実装し、基本的なスレッド操作の処理時間を測定した。その結果、別のモジュール上へのスレッドの生成は、同一モジュール上での操作の2.7倍の処理時間で実現できた。また、あるタスクセットを用いた評価では、モジュール間通信の割合は50%減少し、外界からの入力に対する反応時間は8.0%短縮された。

Dividing a Task and Executing it in Parallel on a Function-Classified Parallel Computer

MITSUHIKO OHTA,† MICHIMUNE KOHNO,† TOSHIYUKI MAEZAWA† and YUICHIRO ANZAI†

In a functionally classified parallel computer, tasks which are functionally divided are both programmed and executed in parallel in each modules. But these operations are difficult. In this paper, we propose a system that divides a user task which includes some threads into some functional modules at a run time and executes these divided tasks in parallel. In this system, a thread which frequently accesses a particular device is executed in the module which has that device. We consider that this system can reduce inter-module communications between server threads which control the device and user threads.

We implemented this system into *μ–PULSER*, an operating system for personal robot. We evaluated some basic thread operations in this system. The result shows that the cost of creating a remote thread is 2.7 times as long as that of creating a local thread. And we also evaluated the rate of inter-module communications. The result shows that it was reduced to 50% and that the response time of a event is reduced to 8.0%.

1. はじめに

我々の研究室ではパーソナルロボット用アーキテクチャ*ASPIRE* を開発している[8)9)]。*ASPIRE* はセンサ制御やモータの制御などの各機能ごとにハードウェアモジュールを構成し、それらのモジュールが協調して動作する機能別並列計算機という形態をとっている。各モジュール上には パーソナルロボット用 OS *μ–PULSER* が稼働し、機能分散されたタスクが並列に実行される。全てのモジュール上で *μ–PULSER* が稼働しているので、どのモジュール上でもユーザタスクを実行させることが可能である。しかしこれまで、ユーザタスクはひとつのモジュール上のみで並行に実行されることが多く、機能別並列計算機を有効に活用していなかった。これは、複数のモジュール上で別々にユーザタスクを実行させることが困難であったためと考えられる。

そこでひとつのタスクを、各機能別モジュール上で実行させる複数のタスクに分割し、それらのタスクを並列実行させる機構を *μ–PULSER* 上に実装する。この機構を使用することによって、これまでユーザがモジュールごとに分割して記述していたタスクをまとめてひとつのユーザタスクとして記述でき、*ASPIRE* の持つ利点を活かしたユーザタスクの実行が容易にできるようになると考える。さらに、分割したタスク間で共有される変数を、ひとつのモジュールで実行するタスクの記述と同様に使用することができるように実装

† 慶應義塾大学
Keio University

する。

この機構はタスクの生成時に、特定のデバイスを頻繁にアクセスするスレッドをそのデバイスが存在するモジュール上で実行させるようにタスクを分割する。これによりユーザタスク中のスレッドが、モジュール内で行う通信回数よりも別モジュールとの通信回数の方が多いという問題[6]を解決することができると考える。

この機構を用いた基本的なスレッド操作の処理時間を測定した。その結果、別のモジュール上へのスレッドの生成は、同一モジュール上での操作の2.7倍の処理時間で実現できた。また、あるタスクセットを用いた評価では、モジュール間通信の割合は50%減少し、外界からの入力に対する反応時間は8.0%短縮された。

以下、第2章では本研究の実装対象について、第3章で本研究の目的を述べる。また、第4章でタスク分割機構の設計と実装について述べ、第5章で評価及び考察を行なう。最後にまとめと今後の課題を述べる。

2. 背　　景

本章では実装対象のパーソナルロボット用アーキテクチャ *ASPIRE* とパーソナルロボット用 OS *μ-PULSER* について説明し、現在の*μ-PULSER* の持つ問題点について述べる。

2.1 パーソナルロボット用アーキテクチャ *ASPIRE*

ASPIRE はバス結合型の機能別並列計算機として実装されている。センサ、モータなどのロボットに必要な要素を機能ごとに個々のハードウェアモジュールとして構成し、それらを VME バスを用いて結合しており、次のような特徴を持つ。

- 各モジュールは機能の異なるデバイスを持つ
- 機能分散されたタスクを並列実行する

ロボットに搭載される機能別モジュールのうち、ロボット全体の統括を行うものをメインモジュールとし、これがプランニングやロボットの位置特定などの高度な処理を行う。また、単体でも動作できる程度の処理能力を持つモジュールがデバイス制御を行う。このようにすることによって、様々な用途に使用することが可能なロボットを構築することができると考えられる。また、モジュール間の同期や I/O などのイベント伝達に割り込みを用いることによって、外界からの入力に対して迅速に対応できるハードウェア構成を提供している。例えば、センサによって障害物を発見してからモータを停止するといった動作は、ロボット全体の統括を行うメインモジュールを介すことなくセンサモジュールから直接モータモジュールにイベントを伝達することによって実現できる。

2.2 パーソナルロボット用 OS *μ-PULSER*

ASPIRE をサポートするために、我々はパーソナルロボット用 OS *μ-PULSER* を開発している。*μ-PULSER* の基本的な機能は、マイクロカーネルと複数の管理スレッドによって提供される[7]。

また、ソフトウェア割り込みとハードウェア割り込みを統合して扱うコンカレント・インタラプト (以降 CI と記す) 機構によって、リアクティブ性を実現している。さらに、リアルタイムスケジューラによって時間制約を持つスレッドの管理が可能である[4]。これにより、モータやマニピュレータの制御などを行うスレッドの実時間制約を保証することができる。

2.3 *μ-PULSER* の問題点

全てのモジュール上で *μ-PULSER* が稼働しているので、どのモジュール上でもユーザタスクを実行させることが可能であるが、以下の理由からこれまで複数のモジュール上にユーザタスクを並列実行させることがで困難であった。

- モジュールごとに分割したタスクの記述が困難
- 各モジュール上でタスクを実行させる操作が繁雑

このため、ユーザタスクはメインモジュール上のみで並行に実行され、各機能別モジュールがメインモジュール上から使用されるという、機能別並列計算機に適していないタスクの実行形態がとられることが多かった。またこのようなタスクの実行形態がとられた場合、VME バスを介したメッセージパッシングによって各機能別モジュールの持つデバイスにアクセスすることになるため、別モジュールへの通信量が多くなる。*ASPIRE* は VME バスを使用しているため、バスの使用率が高くなるとタスクの実時間制約を保証できないという問題があり[6]、モジュール間の通信量が多いことが問題であった。

2.4 関連研究 : PVM

PVM(Parallel Virtual Machine) は、ネットワークで接続された異機種の計算機を、単一の大きな並列計算資源に統合するパッケージソフトウェアである[2]。PVM は pvmd と呼ばれるデーモンと、プロセス間通信・プロセス操作・タスクの協調などの機能を提供するユーザライブラリ（PVM ライブラリ）から構成される。

ユーザは C 又は FORTRAN を用いてアプリケーションを記述する。その際に、使用する各計算機上で共通に使用できるメッセージパッシングライブラリを利用する。アプリケーションを構成する複数のタスクは協調して動作し、並列に計算を行うことができる。

PVM は異機種の計算機の利用をサポートしている。このため PVM を利用するタスクは各計算機上に各アーキテクチャごとの実行形式ファイルを用意しておく必要がある。各計算機上に置いた実行形式ファイルを起動し、実行したい計算機上のプロセスを呼び出すことによって並列にプロセスを実行する。

本研究では PVM を参考にして、別モジュールからのスレッド操作を代行するサーバを *μ-PULSER* に追加し、スレッド操作のためのライブラリを変更する。

そして、メッセージパッシングを用いて別モジュールへのスレッド操作を行えるように設計する。

3. 目　　的

本研究の目的は、各モジュール上で実行させるスレッドをひとつのユーザタスク中に記述し、実行時にモジュールごとに分割して並列実行できるような機構を設計し、それをパーソナルロボット用 OS *μ-PULSER* 上に実装して評価することである。

ユーザタスクと各モジュール上のサーバとの通信量を減少させるため、スレッドが頻繁にアクセスするデバイスの存在するモジュール上で実行されるようにタスクを分割する。

4. 設計及び実装

本機構の追加・削除を容易にするため、各モジュール上に起動するサーバの追加とライブラリの変更だけで済むように設計した。

4.1 タスクの操作

タスクはメインモジュール上にロードされた後に他の全てのモジュール上に転送される。そして、メインモジュール上のみで `main()` 関数を実行し、その他のモジュール上ではタスクの初期化のみを行う。このようにタスクを生成時に全てのモジュールに転送しておくことにより、スレッド生成のオーバヘッドを削減でき、特にスレッドの生成に実時間制約が要求される場合に有効であると考える。

4.2 スレッドの操作

本研究ではスレッドを、頻繁にアクセスするデバイスの存在するモジュール上で実行させることによって、モジュール間の通信量の削減を図る。そのために、`thread_create()` 関数の引数に頻繁にアクセスするデバイスに対応したデバイスナンバを明示的に指定させる。このデバイスナンバからスレッドを生成するモジュールを判定する。関数を呼び出したモジュール上にスレッドを生成する場合は、そのモジュール上のタスク・スレッド管理スレッド (以降 ttm と記す) にスレッド生成を要求する。別モジュール上にスレッドを生成する場合には生成するモジュール上の並列タスク管理スレッド (以降 ptkm と記す) に対して、スレッド生成を要求する（図 1)。

この関数の返り値は生成したスレッドのスレッド ID であり、図 2のような構造になっている。スレッドの開始・停止・削除といったスレッド生成以外のスレッドの操作では、このスレッド ID から操作を行うスレッドの存在するモジュールを判定し操作要求を行う。

4.3 共 有 変 数

ひとつのタスクが複数のモジュール上に分割されるため、タスク中の共有変数は分割されたタスク間でも共有される必要がある。このため、ユーザタスク中の

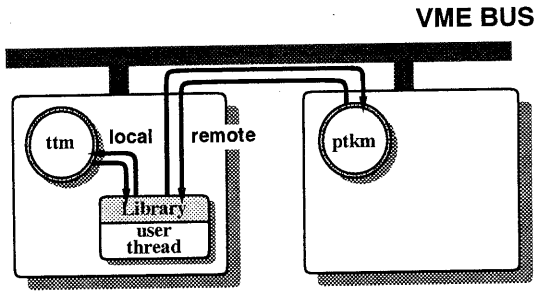

図 1　スレッド操作の要求

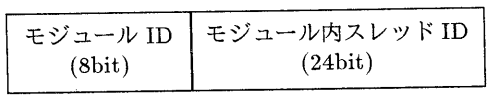

図 2　スレッド ID の構造

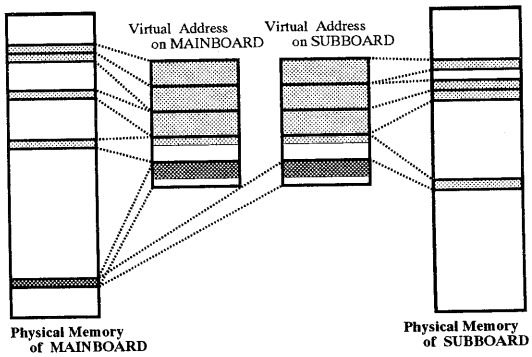

図 3　ユーザタスクのマップ

共有変数のアドレスは全てのモジュールでひとつのアドレスを参照するように、同じ分散共有メモリを指すようにする。図 3に示すようにタスクの生成時に、共有変数領域がその他の領域とは別のページになるように分散共有メモリにマップする。

4.4 実 装 環 境

使用するモジュールは、MC68030(25MHz) をプロセッサとする AVME-130-1 ボード (RAM 8MB, ROM 2MB, NVRAM 128KB, シリアルポート × 2, パラレルポート × 1, イーサポート × 1) である。VME バスインタフェースを持ち、バスの調停機能を持つ。RAM はデュアルポートメモリであり、どのモジュールからも読み書きが可能である。

本論文では以降、イーサポートを使用するモジュールをメインモジュールと呼び、イーサポートを使用しないモジュールをサブモジュールと呼ぶことにする。

5. 評価及び考察

本章では、スレッド操作にかかる時間を測定する。また実装したタスクの分割機構をタスクセットを用い

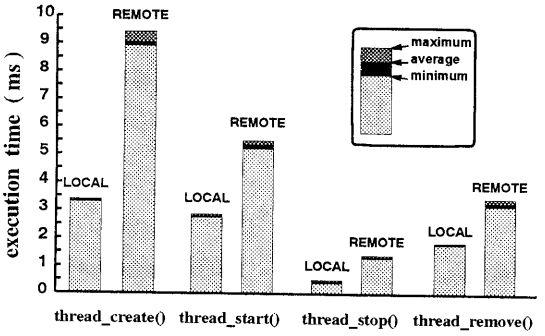

図4　スレッド操作の処理時間

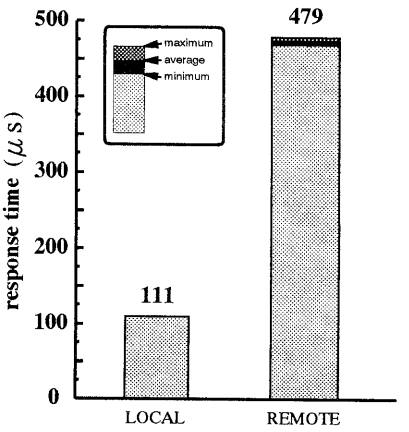

図5　スレッドの反応時間

て評価する。

実行時間の測定はメインモジュール上のカウンタを用いて、1μs 単位で行った。

5.1　スレッド操作にかかる時間

まずスレッドの操作にかかる時間を測定した。スレッドの生成・停止・削除は、スレッド操作関数が呼び出されてから終了するまでの時間を、スレッドの開始は関数が呼ばれてからスレッドが開始するまでの時間を、それぞれ同一モジュールに対する操作 (LOCAL) と別モジュールに対する操作 (REMOTE) に関して測定した。

測定は、メインモジュール上にスレッドが22個、サブモジュール上に21個存在する状態で行った。測定はメインモジュール上に41個、サブモジュール上に35個のエントリがある状態で行った☆。この状態でそれぞれの処理時間を500回ずつ測定した。測定結果を図4に示す。

図4に示すように、LOCAL に対して REMOTE にかかる時間は2倍から3倍大きい。主な理由は、別モジュールへスレッド操作を要求するためにメッセージパッシングを用いており、その処理時間が長いためである。さらに、ttm に対して送る情報よりも、別モジュール上の ptkm に送る情報の方が多いことや、モジュール間でのタスク ID の変換処理など、別モジュールに操作を要求する方がより多くの操作を必要とするためである。

しかし、別モジュール上に多くのスレッドを生成する必要がある場合には、あるひとつのスレッドを別モジュール上に生成し、そのスレッドから多くのスレッドをそのモジュール上に作成することによって、モジュールをまたがったスレッド操作を減らすことができると考える。

5.2　スレッドの反応時間

μ-PULSER では、スレッドの起動やイベントの伝達を CI という割り込みを用いて行っている。この CI の発生からスレッドが起動されるまでにかかる時間を測定した。測定は、あるスレッドから別のスレッドに CI を発生しスレッドが起動するまでの時間を、ふたつのスレッドが同一モジュール上にある場合（LOCAL）と別モジュール上にある場合（REMOTE）に関して行った。測定環境は、スレッド操作の評価と同様である。それぞれを1000回ずつ行った結果を表5に示す。

CI が発生してからスレッドが起動するまでの時間は、同一モジュールのスレッドの起動に比べ、別モジュール上のスレッドの起動には約4.3倍の時間がかかっている。この原因のひとつとして VME バスを介した CI の発生があげられる。VME を介して別のモジュール上のスレッドに CI を発生した場合は4.3倍、時間差にして約400μs 多くの時間を要する。別モジュール上の ptkm への処理依頼にはメッセージパッシングが使用されており、この送受信で合計2回の VME を介した CI が発生する。また、VME バスを介したメモリのコピーにかかる時間は同一モジュール内におけるメモリのコピーよりも多くの時間を要する。

この VME バスを介した CI の発生は、モジュールをまたがる全てのスレッド操作やスレッド間の同期に使用されるため、これらの処理時間に影響を及ぼす。したがって、ロボットの即応性に大きな影響を与えると考えるため、この CI の処理を高速化することが重要であると考える。本研究では、この VME バスを介した CI の発生機構としてこれまで *μ-PULSER* に実装されていた機構をそのまま用いた。

VME バスを介した CI の処理を高速化するためには、カーネル内にある CI ハンドラが別モジュール上のスレッドを起動するための CI であった場合に直接起動するスレッドが存在するモジュールに VME 割り込みをかけるようにすればよいと考える。

5.3　タスクセットを用いた評価

実装した機構が、モジュール間の通信回数と外界の変化に対する反応時間に与える影響を評価するために、以下のようなタスクセットを用いる。

☆ CI の発生にかかる時間は CI のエントリ数に比例する。

モジュール固有のデバイスを管理するサーバとして、モータを管理するもの（SERVER_1）と、センサを管理するもの（SERVER_2）を起動する。そして、SERVER_1 に頻繁にアクセスするスレッド th 1 と、SERVER_2 に頻繁にアクセスする th 2 を含むユーザタスクを起動する。th 2 がセンサデータによってロボットの動作を決定し、th 1 がモータの駆動要求を行うことによって、障害物を回避しながらロボットが動作するというタスクを評価に用いた。

図6に示すように、ユーザタスクがひとつのモジュール上にある場合をモデル A、実装した機構を用いてふたつのモジュールに分割した場合をモデル B とする。

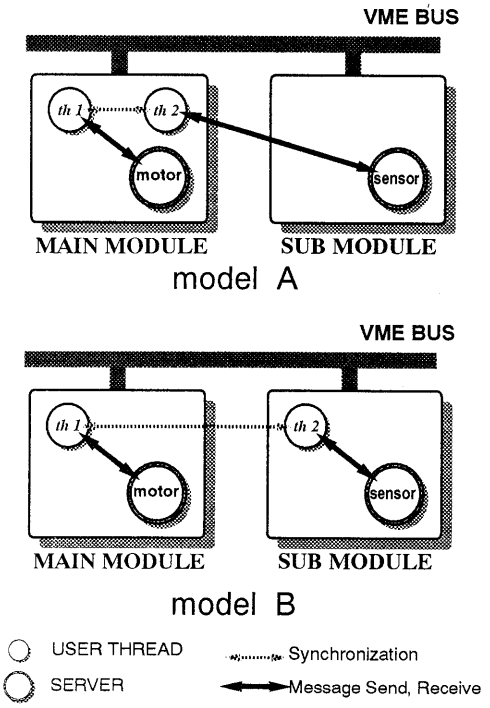

図 6　タスクセット

5.3.1　モジュール間通信の回数

このタスクセットを用いてユーザタスクと各種サーバが行った通信の回数を測定した。結果を、メッセージパッシング・同期（ CI の発生）それぞれについて、LOCAL と REMOTE に分けて示す（図 7）。また、図 7における凡例は、色の濃いものほど処理時間がかかる。

この結果から、デバイスを管理するサーバへの要求を、タスクを分割しデバイスの存在するモジュール内の通信で行うことにより、バスを介した通信（図 7中の REMOTE）の量が減少したことがわかる。このメッセージパッシングにかかる処理時間を表 1 に示す。VME バスを介したメッセージパッシングと CI を

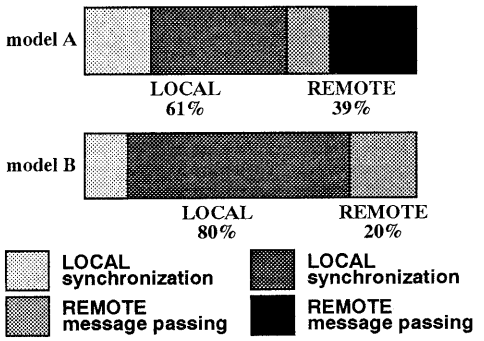

図 7　ユーザタスクの行った通信回数

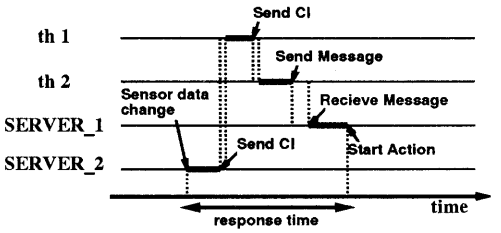

図 8　反応時間の測定

用いた同期操作による通信を比較すると、メッセージパッシングを用いた通信では、メッセージのヘッダのコピーの分だけ多くバスを使用する。このため、VME バスを介したメッセージパッシングの回数を削減することによって、バスの使用量が減少する。

表 1　メッセージの送受信にかかる時間

	最小値 (μs)	最大値 (μs)	平均値 (μs)
LOCAL	543	546	544
REMOTE	1227	1238	1232

5.3.2　外界の変化に対する反応時間の測定

タスクセットを用いて、外界の変化に対するパーソナルロボットの反応時間として、図 8に示す時間を測定した。この測定をモデル A とモデル B について 100 回ずつ行った結果を表 2に示す。

表 2　外界の変化に対する反応時間

モデル	最小値 (μs)	最大値 (μs)	平均値 (μs)
model A	1735	1763	1741
model B	1527	1617	1617

この結果は、実装した機構を用いてタスクの分割を行った場合（モデル B）の方が約 8.0%イベントの伝達時間が短縮されたことを示している。このことから、モジュールにまたがったイベントの伝達時間が速くなったことがわかった。

モデル A とモデル B を比較すると、VME バスを

介した通信にメッセージパッシングを用いているモデルAよりも、共有変数に値を書き込み割り込みをかけるだけのモデルBの方が、メッセージパッシングに用いられるメッセージのヘッダのコピーにかかる時間の分だけ伝達時間が短縮されている。

6. まとめ

本研究では、複数のモジュール上で実行させるスレッドを含むひとつのユーザタスクを、実行時にモジュールごとに分割し、並列実行させる機構を設計し、パーソナルロボット用OS *μ-PULSER* 上に実装した。この機構によって、これまでユーザがモジュールごとに分割していた複数のタスクを、ひとつにまとめて記述することができるようになった。

ロボットを制御する上で必要とされるようなタスクモデルを用いた評価の結果、使用するデバイスを考慮してタスクを分割することによって、モジュール間の通信量が50%減少することがわかった。

しかしVMEバスを介した割り込み機構が複雑なために、別モジュールへのスレッドの操作にかかるオーバヘッドが、同一モジュール上での操作に比べて2倍から3倍大きいという問題があり、この割り込み処理を高速化する必要があることが分かった。

今後の課題として、現在移植中のSPARC版の *μ-PULSER* にこの機構を適用したいと考えている。

参考文献

1) Adam Ferrari and V. S. Sunderam: "TPVM: Distributed Concurrent Computing with Lightweight Processes", Technical report, Dep. of Math. and Computer Science, Emory University (1994).
2) Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidy Sunderam: *PVM 3.0 USER'S GUIDE AND REFERENCE MANUAL* (1993).
3) Hondo Zhou and Al Geist: "LPVM: A Step Towards Multithread PVM", http://www.epm.ornl.gov/~zhou/lpvm.ps (1995).
4) 飯田浩二, 矢向高弘, 菅原智義, 安西祐一郎: "スケジューリングポリシの動的変更に関する研究", 実時間処理に関するワークショップ(RTP'93) 情報処理学会研究報告 93-ARC-99, pp. 70–76 (1993).
5) Jeremy Casas, Ravi Konuru, Steve W. Otto, Robert Prouty and Jonathan Walpole: "Adaptive Load Migration systems for PVM", *Proceedings Supercomputing '94*, pp. 390–399 (1994).
6) 河野通宗, 前沢敏行, 安西祐一郎: "単一仮想記憶空間を提供するリアルタイムOSの設計", 情報処理学会システムソフトウェアと オペレーティングシステム研究会報告, pp. 33–38 (1995).
7) 菅原智義, 飯田浩二, 秋庭朋宏, 紺田和宣, 矢向高弘, 安西祐一郎: "パーソナルロボットのための敏感で柔軟なOS *μ-PULSER* の設計と実装", 日本ソフトウェア科学会第9回大会論文集, pp. 253–256 (1992).
8) Nobuyuki Yamasaki and Yuichiro Anzai: "Active Interface for Human–Robot Interaction", *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 3103–3109 (1995).
9) 山崎信行, 安西祐一郎: "パーソナルロボット用機能別並列計算機アーキテクチャ: *ASPIRE*", 情報処理学会論文誌, pp. 81–91 (1996).