ソフトウェア工学 120－13
（1998．7．24）

実行モデルに基づく並列プログラミング支援環境の構築

和泉秀幸† 中島毅†

本稿では，実行モデルに基づく並列プログラム開発支援環境の構築法について述べる．この方法では，アプリケーション開発者が必要とする並列処理機構を“実行モデル”として定め，支援環境側がマルチスレッドライブラリとして提供する．支援環境は提供したI/Fを使ったプログラムを対象に，作成からデバッグ・性能評価までの一連の開発作業を支援する．これにより，一般に困難といわれるマルチスレッドプログラムの開発が容易にでき，かつ品質の高い開発が可能となる効果がある．

本方法の効果を検証するため，監視・制御システムの操作端末S/Wを対象に支援環境を適用した．対象S/Wは同じ入力機器から異なるリアルタイム時間制約条件の処理が投入される特徴を持つ．これに対して“ディスパッチ実行モデル”を規定し，簡易スレッドライブラリと性能評価支援ツールを実現した．これらを使って開発を行い，プログラム作成と性能評価の負荷を削減できることを確認した．

Execution Model base Multithread Programming Support Environment

HIDEYUKI IZUMI† and TSUYOSHI NAKAJIMA†

This paper describes a Parallel Programming support Environment based on execution models (we call it M-PPE). When we develop a multithread program, application programmers have to implement a parallel control mechanism for each application. We define this mechanism as “execution model”. In M-PPE, the execution model and its mechanism are implemented for each target program. Programmers make application using M-PPE I/F. M-PPE supports a sequence of development (from creating program to debugging/tuning) of applications using its I/F.

We have implemented M-PPE for an application that controls a user interface terminal on real-time system. We have defined a “Dispatch Execution Model(DEM)” and, created a Dispatch library and tuning support tools for DEM. Using these tools for developing an application, we could reduce the cost of developing a multithread program.

1. はじめに

並列プログラムの作成方法の1つに，スレッドライブラリを利用したマルチスレッドプログラムがある．近年では，WSやPCといった汎用の環境で，マルチスレッドプログラムの開発が可能になってきている．しかし，一般に，正しくかつ効率の良いマルチスレッドプログラムを開発することは難しいため，プログラミング支援環境に対する期待が大きい．この支援環境には，次のことが求められる．

- 開発サイクルを支援
  並列プログラムは，並行処理の実現手段として利用され，高速化に役立つ他，信頼性向上にも応用される．このため，目的の機能と性能が得られるまで“作成，デバッグ，性能評価”といった一連の開発サイクルを繰り返すが，この全工程にわたって支援できること．
- デバッグ・性能チューニング支援
  並列プログラムでは同期や共有資源へのアクセスタイミングが，動作，性能，実行結果に影響を与える．このタイミングを評価できること．
  また，並列プログラムには，実行順序の再現性がない，実行可能な順序のパターンが爆発的に増加する，デッドロックの発生などの従来の逐次プログラムにはない問題がある．このような問題に対処する機能を実現すること．
- 並列処理機構の実現支援
  システム提供の並列プログラム作成I/Fを使って目的のアプリケーションに適した並列処理機構の実現を支援すること．この並列処理機構には，アプリケーションを分割し，分割した処理の依存関係を維持しつつ効率良く実行するように割り振る機能や，同期など共有資源へのアクセス制御機能が含まれる．

このような支援環境は，支援対象プログラムの作成方法で大きく分類することができる．1つは，自動並列化コンパイラなど使って“逐次プログラムから自動的に並

† 三菱電機株式会社
Mitsubishi Electric Corporation

列実行モジュールを生成する" 方法[9]. もう1つは, 開発者がシステムが提供するI/Fを使って"プログラム内で明示的に並列処理を記述する" 方法である. 今回我々が対象としたのは, 開発者がスレッドライブラリを使用して"明示的に並列処理を記述する" 方法でアプリケーションを作成する支援環境に関してである.

"明示的に並列処理を記述する" 方法では, 並列記述言語によりスレッド生成や通信機能などを隠蔽することでユーザの負担を軽減する手法が提案されている[6]. また, 制御フローの可視化を利用して並列処理を実現する環境も提案されている[7]. これらは, 計算機依存部分を吸収し, 移植性を高める手段としても利用されている.

この他, マルチスレッドプログラムの開発に必要な基本的機能を備えた統合的な支援環境としてSUN社のSPARC Works[3][4][5], ENCORE社のParasight[1][2]がある. SPARC Worksは, ソフトウェア開発の統合環境であり, デバッガと性能評価ツールの他, 静的なプログラム検証ツールがある. ENCORE社のParasightは, デバッガと性能評価ツールであり, デバッグを行いながら性能評価を効率良く行える特徴がある.

このような並列記述言語や統合的な支援環境は, 汎用性を重視しており, 様々な種類の並列処理機構を採用したアプリケーションで利用できる. このため, 最適の並列処理機構を選択するために, 様々な実現方法を切り替えながら評価する時に有利である.

一方, 我々は並列処理機構を限定することで, 開発効率の良い支援環境が構築できると考える. まず, 性能評価とデバッグの視点からは, 汎用性を重視した方法よりも収集対象の情報を絞ることができ, プログラムの実行情報を効率良く収集できる. 例えば, 並列プログラムの再現実行には, "実行順序に影響を与える箇所の実行順序と制御情報" を記録し, 再現する方法がある[8]. この"実行順序に影響を与える箇所と制御情報" は適用する並列処理機構ごとに異なるので, 全スレッドプログラムを対象に収集するのは難しい. 並列処理機構を限定することで, 実行順序に影響を与える箇所を特定でき, 情報の入手が容易になる.

また, プログラム作成の視点からは, 次のような効果がある. 大規模なプログラム開発では, 複数の人が並列処理の制御をそれぞれ独立に変更すると, ロックの獲得・開放の不整合などを引き起こし, 信頼性を低下させる可能性が高くなる. このような開発では, 並列処理の制御方法に一定の枠組を与えることで, 開発者間での不整合を効率良く減少できる.

これらのことから, 我々は開発対象のアプリケーションに適した並列処理機構を定め(本稿では, この並列処理機構を"実行モデル" と呼ぶ), 定めた"実行モデルに基づく並列プログラミング支援環境" M-PPE(Model based Parallel Programming support Environment) を構築している.

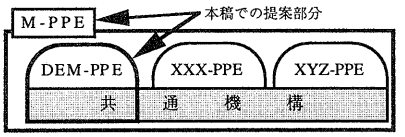

図1 実行モデルに基づく支援環境

我々は, M-PPEの効果を確認するため, 実システムへの適用を進めている. 本稿では, 実際にリアルタイムアプリケーションに適用した例を示す. 適用対象アプリケーションは, 監視・制御システムの操作端末のソフトウェアであり, 同一の入力機器から異なった時間制約条件が課せられた処理を投入される特徴がある. 今回我々は, このソフトウェアに適した並列処理の実行モデルを"ディスパッチ実行モデル" DEM(Dispatch Execution Model) として規定した. DEMでは, "ディスパッチテーブル" と呼ぶ一時記憶を介して"入力受付" と"対応する処理実行" を別々のスレッドで非同期に実行する.

このDEMに基づく支援環境として"ディスパッチ実行モデルに基づく並列プログラミング支援環境" DEM-PPE(Dispatch Execution Model based Parallel Programming support Environment) を構築した. DEM-PPEでは, 簡易スレッドライブラリの"ディスパッチルーチン" と性能評価支援ツール群を開発した. これらを使って実プロジェクトでのソフトウェア開発を支援することで, DEM-PPEの効果を検証した.

本稿では, まず2章で実行モデルに基づく支援環境について説明する. 次に, 3章では適応対象のアプリケーションとその並列実行モデルであるDEMについて述べ, 4章でDEMに基づいて構築したDEM-PPEについて説明する. 5章では考察を行う.

2. 実行モデルに基づく支援環境

ここでは, "実行モデルに基づく並列プログラミング支援環境" について述べる.

並列処理機構を限定することで, 並列プログラムの性能評価やデバッグを有利にする. 例えば, "再現実行情報の収集問題" がある. 並列プログラムは基本的に実行順序の再現性がないため, そのデバッガは再現実行機能が求められる. 再現実行方法としては, "実行順序に影響を与える箇所の実行順序と制御情報" とOS I/Fなどの入出力環境を記録し, 再現する方法が一般的である[8]. この"実行順序に影響を与える箇所と制御情報"

は，適用する並列処理機構ごとに異なる．このため，汎用的に利用できる収集方法の実現は難しい問題がある．これに対して，並列処理機構を限定することで，実行順序に影響を与える箇所を特定できる．特定した箇所で情報収集する機能を支援環境が実現することで，“再現実行情報の収集問題”を改善できる．

他の例としては，“情報収集により実行タイミングが影響を受ける問題”がある．並列プログラムの性能評価では，同期や共有資源へのアクセスの実行タイミングが重要である．しかし，情報収集にかかる時間により，評価対象の実行タイミングが影響を受ける問題がある．これに対して，並列処理機構を限定することで，評価で重要な情報を絞り込むことができる．重要な情報だけを効率良く収集し，“実行タイミングが影響を受ける問題”を極力抑えることができる．

並列処理機構を限定することは，プログラム作成作業の効率化にも効果がある．例えば，大規模なプログラム開発では，複数の人が並列処理の制御をそれぞれ独立に変更すると，ロック獲得・開放の不整合などをおこし易い．これは，プログラムの信頼性や性能を低下させるが並列処理機構を限定して制御方法に一定の枠組を与えることで，開発者間での不整合を減らすことができる．

また，並列プログラムの開発経験が少ない開発者には，対象アプリケーションに適した並列実行方法の選択が難しい．支援環境が並列処理機構をを実現し，簡易スレッドライブラリとして開発者に提供されていれば，並列処理やスレッドライブラリに対する経験が少ない開発者の負荷を軽減できる．

これらのことから，我々は開発対象のアプリケーションに適した並列処理機構を実行モデルとして定め，それに基づく並列プログラミング支援環境 M-PPE を構築している．

M-PPE は次のように構築する．

- 開発対象のアプリケーションに適した並列処理機構を実行モデルとして規定．
- 規定した並列処理機構を簡易ライブラリで実現．
- 実現した簡易ライブラリを使用したアプリケーションを対象に，デバッグやチューニングの支援を行うツール群を開発．

開発者は M-PPE 提供のライブラリ I/F を利用することで，並列処理の制御に関する手続きをプログラム中に記述せずに，アプリケーションに適した並列処理機構を実現できる．また，M-PPE で規定した実行モデルを複数の開発者間での共通基盤にすることで，ロック獲得・開放の不整合などを減少できる．

M-PPE では自身が提供した I/F を利用して開発したアプリケーションを対象に，性能評価を行う．このため，性能評価やデバッグを行うツールでは，対象の実行モデルでポイントとなる計測箇所と情報を明確にし，必要な情報を自動的に収集できる．また，自動収集した情報を使って効率の良い性能評価や，再現実行機能などを実現することが期待できる．

3. ディスパッチ実行モデル

この章では，並列プログラミング支援環境の適用対象のリアルタイムアプリケーションについて説明した後で，規定した実行モデルである“ディスパッチ実行モデル”について述べる．

3.1 対象のリアルタイムアプリケーション

ここでは，適用対象のリアルタイムアプリケーションを説明する．対象とするリアルタイムアプリケーションは，次のような機能を持つ監視・制御システムの操作端末のソフトウェアである (図2参照)．

- システム内で採取した情報を表示
  システムからの状況変化通知やユーザの端末への操作をトリガーとして，制御システム内の情報を画面に表示する．
- ユーザ操作をシステムへの制御命令として発行
- 各処理にリアルタイム時間制約が設定
  画面への表示や制御命令の発行などの処理ごとに，異なったリアルタイム時間制約が仕様として設定されている．

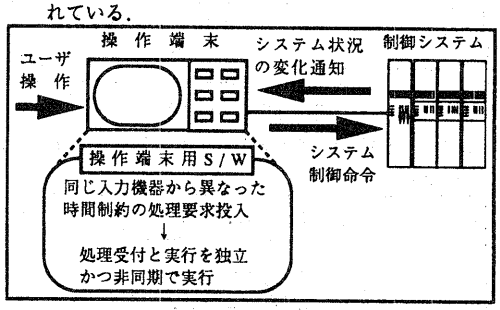

図2 監視・制御システムの操作端末のソフトウェア

この監視・制御システムの操作端末のソフトウェアには次のような要求事項がある．

- アプリケーションへの要求
  監視・制御システムの操作端末では，同じ入力機器から様々な処理要求が投入される．これらの処理要求には，それぞれ異なった時間制約条件が指定されている．対象アプリケーションは，各入力機器ごとに処理を受け付ける．このため，アプリケーションが (1) 入力受付と (2) 入力に対応する処理の実行を，独立かつ非同期で実行できること．

3.2 構成要素と実行手順

3.1 節で示したようなアプリケーションに対する実行モデルとして，我々は“ディスパッチ実行モデル” DEM(Dispatch Execution Model) を規定した．この節では DEM の構成要素と，並列処理の実行手順について説明する．

まず，DEM の構成要素を示す (図 3 参照)．

- 登録スレッド
  外部入力を監視していて，入力時に対応する処理と優先度をディスパッチテーブルに登録する．
- 実行スレッド
  ディスパッチテーブルを周期的にチェックし，登録された処理を優先度に従って実行する．
- ディスパッチテーブル
  処理と優先度を一時的に記憶するテーブル．

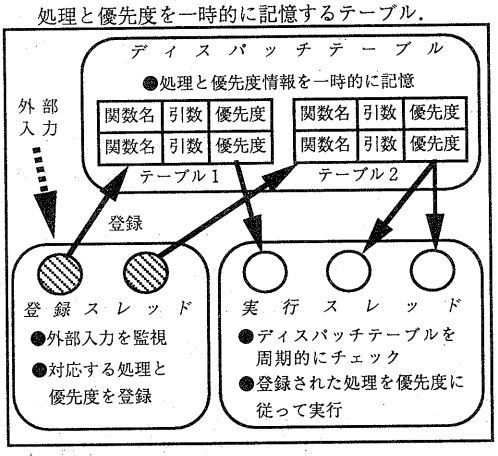

図 3 DEM の構成

次に，DEM での処理の基本的な実行手順を説明する (図 4 参照)．

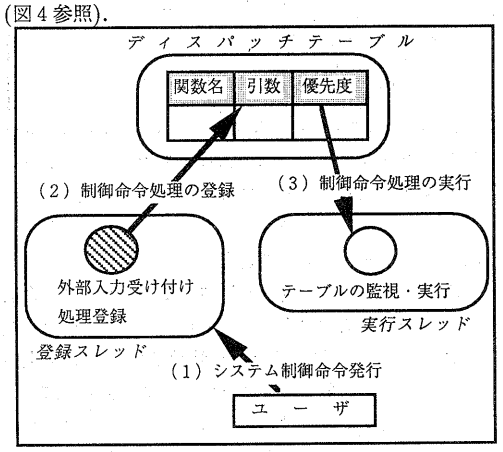

図 4 DEM の基本的な実行手順

まず，登録スレッドが外部入力を受け取る．各登録スレッドには，監視対象の外部入力が指定されており，図 4 の例では，“システム制御命令発行” に対応するスレッドが受け取る．次に，外部入力に対応する処理を，DEM-PPE の I/F (4 章参照) を使ってディスパッチテーブルに登録する．この例では，“システム制御命令発行” に対応する関数をテーブルに登録する．

実行スレッドは登録スレッドとは非同期に，ディスパッチテーブルを一定の周期でチェックし，登録済み処理があればプライオリティに従って実行する．図 4 の関数はテーブルに登録済みとなった後で，実行スレッドがチェックを行うタイミングで実行される．

このように DEM では，外部機器からの入力を受け付ける“登録スレッド” と，その入力で要求された処理を実行する“実行スレッド” が別々のスレッドで実現され，それぞれ非同期に実行される．このため，入力受付と対応する処理の実行が独立かつ非同期で実行できる．

4. プログラミング環境の構築

この章では，“ディスパッチ実行モデル” に基づく“並列プログラミング支援環境” DEM-PPE (Dispatch Execution Model based Parallel Programming Support Environment) の構築について説明する．DEM-PPE では，開発対象の監視・制御システムの操作端末のソフトウェアを DEM に基づいて開発するための基盤を提供することを目的とする．DEM-PPE の構成要素を次に示す．

(1) “ディスパッチルーチン”
DEM に基づく並列実行の枠組を実現し，利用 I/F をライブラリ形式で提供する (4.1 節参照)．対象アプリケーションの実行情報を収集するログ収集機能がある (4.2.2 節参照)．

(2) 性能評価支援ツール
“ディスパッチルーチン” を使って収集した実行情報を解析して，性能評価用の情報を表示するツール群．外部入力単位で処理の所要時間を表示するツールやスレッド毎の実行状況の時系列を表示ツールがある (4.2.3 節参照)．

(3) リアルタイム検証ツール
リアルタイム時間制約を検証するツール．収集した実行情報を使って“外部入力投入から処理実行完了まで” の時間を計算し，リアルタイム制約条件を満たすかどうかを判定する．

(4) デバッグ支援ツール
マルチスレッドプログラム対応デバッガに，DEM 対応の再現実行機能を拡張したツール．

4.1 DEM-PPE でのプログラム作成

DEM-PPE では，開発対象のアプリケーションを“ディスパッチルーチン”を使って作成する．ここでは，ディスパッチルーチンを使用したアプリケーション作成法について説明する．

ディスパッチルーチンは，システム (HP-RT) が提供するスレッドライブラリ (Pthread Library) を利用して，ディスパッチテーブルを使った処理の登録と実行など，DEM に基づく並列処理機構を実現する．実現した並列処理機構は次の I/F 経由で利用する．

- 処理 (関数) を，登録先テーブルと優先度を指定してディスパッチテーブルに登録する I/F
- 実行スレッドでチェックするディスパッチテーブルと優先度の判定アルゴリズムを指定する I/F
- ディスパッチされる処理 (関数) は対象システムごとに異なるため，関数の引数情報を設定する I/F
- ディスパッチテーブルのサイズと ID 設定 I/F

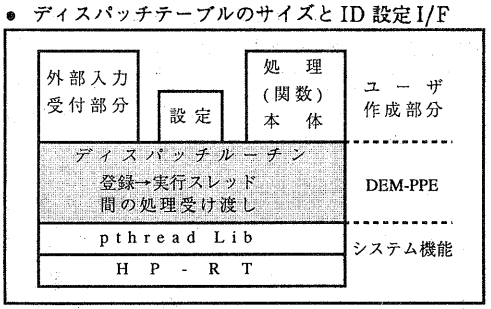

図5 ディスパッチルーチン

図5のようにアプリケーション開発者は，H/W に依存した外部入力の受付部分と，外部入力に対応する処理と，ディスパッチルーチンの設定を実現することで，目的のアプリケーションを得ることができる．ディスパッチテーブルを使って処理をスレッド間で受け渡す部分は，ディスパッチルーチンが行う．

4.2 DEM-PPE での性能評価

ここでは，DEM-PPE での性能評価を説明する．まず，DEM-PPE での性能評価手順を説明する．次に，DEM のアプリケーションで性能評価に有効な情報を示す．その後，DEM-PPE で性能評価情報を収集するログ収集機能と，収集したログ情報から性能評価を行うツールを説明する．

DEM-PPE では，ディスパッチルーチンを使って実現したアプリケーションを対象に性能評価を支援するので，性能評価に有効な情報を絞り込み，その情報を収集する位置をあらかじめ特定できる．このため，DEM-PPE では，まず 1) ディスパッチルーチンのログ収集機能で実行中のプログラムから性能評価情報を自動収集し，次に 2) 自動収集したログ情報を解析して表示するという手順で性能評価を行う．

4.2.1 性能評価に有効な情報

DEM のアプリケーションで性能評価に有効な情報を示す．

- ディスパッチ関連情報
  各外部入力に対する実行タイミングと所要時間情報．処理時間が長い外部入力とその内容 (登録，実行待ち，処理実行) の特定に利用する．実行タイミングは，登録の 1. 開始，2. 完了と，処理実行の 3. 開始，4. 完了の時刻．所要時間は，登録時間，実行待ち時間，実行時間，全所要時間 (図6参照)．

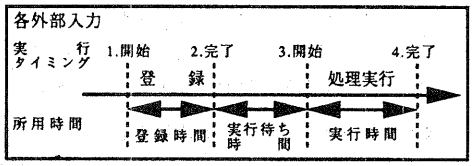

図6 ディスパッチ関連情報

- 時系列データ
  “ディスパッチテーブルに登録中の処理の個数”と“各スレッドの実行状況”の時系列情報．処理時間がかかっている原因の解析に利用する．スレッドの実行状況は，登録中，処理実行中，実行待ち処理無し，CPU 無し，その他 (図7参照)．

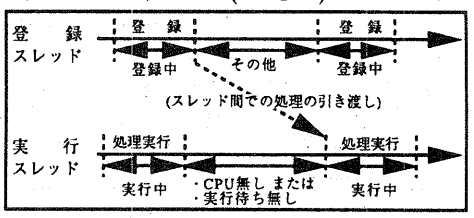

図7 スレッドの実行状況時系列データ

- ロック獲得・開放情報
  ロック獲得と開放のタイミング情報．デッドロック検知やロック待ちによる性能低下の発見に利用する．

4.2.2 ログ収集機能

ディスパッチルーチンには，性能評価に有効な情報を自動収集する“ログ収集機能”がある．この機能は，ログ収集モードで対象アプリケーションを実行した時に，次のログデータを自動的に収集する．

- DEM ログデータ
  ディスパッチ関連情報とスレッドの実行状況時系列情報を得るためのログデータ．各外部入力ごとの，登録の 1. 開始，2. 完了と，処理実行の 3. 開始，4. 完了時刻のログデータ (図6参照)．外部入力識別情報 (4.2.3 節参照) と一緒に収集する．

この他，5. チェック時にテーブルが空，6. テーブルが FULL の時の情報も収集する．

- ディスパッチ・テーブル ログデータ
  テーブルに登録中の処理の個数を得るためのログデータ．個数の変化を時系列で収集する．
- ロック情報
  ロックの 1. 獲得要求，2. 獲得，3. 解放時刻のログデータ．ロックの識別情報と一緒に収集する．

ログ収集によるアプリケーションの実行タイミングへの影響を極力抑えるため，実行中は情報を一旦メモリ中に蓄え，実行終了時にファイルへ書き出すように実現した．このため，実行中にエラーで中断する場合にはメモリ内容をファイルへ書き出す．また，メモリが一杯になった時は内容をファイルへ書き出す．この時，評価時にファイル出力時間を補整するデータを追加する．

4.2.3 ディスパッチ ID 設定機能

ここでは，ディスパッチ ID 設定機能を説明する．まず，DEM 情報のログを収集する時の問題を図 8 を使って示す．

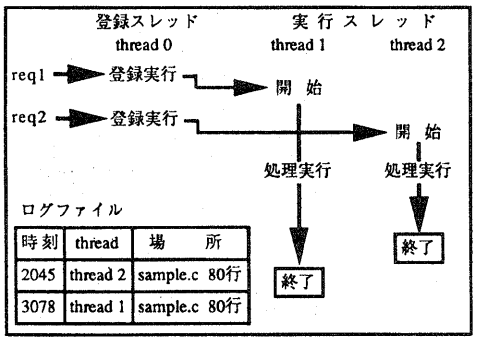

図 8 問題となる例

図中で縦軸方向は時間の経過を，req1 と req2 は同じ種類の外部入力であり，“thread0” はこの外部入力を受け付ける登録スレッド，“thread1” と “thread2” は実行スレッドである．ログファイルは req1 と req2 の処理終了時刻のデータである．

req1 と req2 に対応する処理は，実行スレッド “thread1” または “thread2” が実行する．実行スレッドの選択はプログラム実行中に動的に決定され，実行スレッド間には同期による順序関係が存在しないとする．

この時，実行の順序関係が存在しないため，先に要求された Req1 に対する処理よりも後から要求された Req2 の処理が先に終了することがある．

ログ収集機能では，ログデータとして “実行時刻” と “実行場所” と “実行スレッド ID” を収集する．Req1 と Req2 に対する処理 (実行場所) は全く同じである．また，実行スレッドは動的に決定される．このため，ログデータからは順序が入れ替わったことを判別できない問題がある．

図 8 のような問題を解決するために，我々はディスパッチ ID 設定機能を実現した．ディスパッチ ID 設定機能は，ログ採取時に次のことを行う．

- 各外部入力にディスパッチ ID を設定
- ディスパッチ ID をディスパッチテーブル経由で，登録スレッドから実行スレッドへ処理と一緒に伝達
- ログ出力時にディスパッチ ID を付加

図 9 は，図 8 へのディスパッチ ID 設定機能の適用結果である．図 9 では，ディスパッチ ID 設定機能により req1 と req2 にディスパッチ ID “E1” と “E2” が設定される．ディスパッチ ID は処理と一緒に実行スレッドまで伝達され，ログデータに付加される．

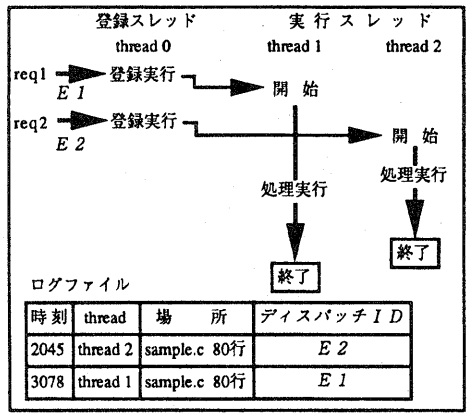

図 9 ディスパッチ ID 設定機能を適用後の例

このように，ディスパッチ ID 設定機能により外部入力とそれに対応する処理を識別できるようにした．

4.2.4 性能評価支援ツール

ここでは，DEM-PPE で実現した性能評価支援ツールを示す．性能評価支援ツールは，ディスパッチルーチンを使って収集した実行ログ情報を解析し，その結果を表示する．次のツールがある．

- タイムログログ解析パーザ
  収集した実行ログ情報を解析して各表示ツールで使用するデータを生成する．

- ディスパッチ実行時間表示ツール
  時間がかかっている場所を特定するためのツール．各外部入力に対する 1. 実行開始待ち時間，2. 処理の実行時間，3. 全実行時間を，時間順にソートしてグラフィック表示する (図１０参照)．

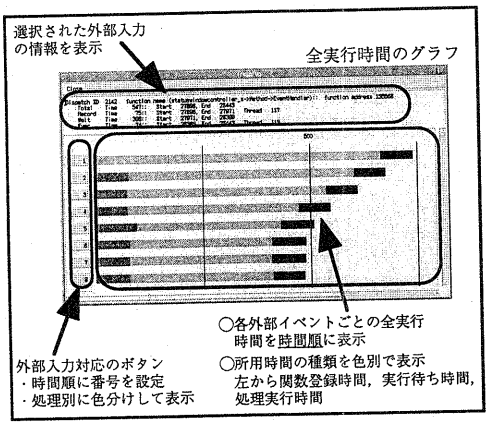

図１０ ディスパッチ実行時間表示ツール

- スレッド実行系列表示ツール
  各スレッドの実行状況を把握するためのツール．各スレッド毎の実行状況を時系列でグラフィック表示する (図１１参照)．

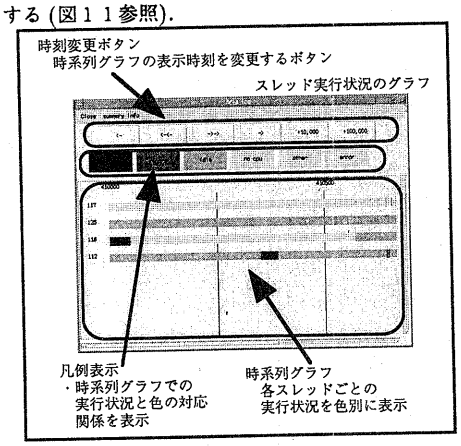

図１１ スレッド実行系列表示ツール

5. 考　　察

ここでは，M-PPE の効果を考察する．まず，実アプリケーションへの適用結果からその効果を検証する．

(1)　プログラム作成支援での効果

- 共通基盤を提供
  従来，オブジェクト指向分析・設計結果から並列処理機構を決定することは難しく，大規模なソフトウェア開発は困難であった．本方式によりいくつかのプロトタイプを試作後，実行モデルを定めることで開発者間の意識を統一することができた．これにより，その後のアプリケーションの各部分の実装設計と開発を進める効果があった．さらに，大人数の開発者間での並列実行制御による不整合を効率良く回避できた．
- 開発者の負荷を低減
  今回の適用例では，各担当者間でのスレッドプログラミングの知識レベルに差異があったが，“ディスパッチルーチン” のように高いレベルでの並列処理機構を提供することで，開発者の負荷をより低減できることが確認した．

(2)　性能評価支援での効果

- 自動ログ収集
  DEM では，アプリケーションで性能評価に重要な情報を収集するポイントを特定することができた．このポイントでの情報収集をディスパッチルーチン内で実現することで，目的の情報を効率良く自動収集できた．また，一般の性能評価にはない，DEM の情報収集で問題となるケースを解決する機能を実現した (ディスパッチ ID 4.2.3 節参照)．
- 解析と表示ツール
  自動的に収集した情報を使って実現したので，解析・表示するログデータに対する知識を利用し，解析と表示ツールを容易に実現できた．また，各外部入力に対する所要時間の情報から処理待ち時間が大きくなる箇所を発見し，プログラムの実行状況の時系列情報を使って原因となる外部入力の登録パターンを見つけることができた．

次に，リアルタイム検証ツールとデバッグ支援ツールへの効果を検討する．今回の適用例では，この 2 つのツールは提案のみで実装していない．しかし，適用支援で実プログラムの実行情報ログデータを収集した．このデータから実現性を検討する．

- リアルタイム検証ツール
  リアルタイム検証ツールの実現には，1. リアルタイム制約条件の記述 (入力) 機能と，2. 判定基準となる実行時間を計測機能と，3. 計測時間と制約条件から判定を行う機能が必要である．DEM-PPE のディスパッチ ID により，各外部入力ごとの全所要時間 (4.2 節参照) が計測可能であり，判定基準の実行時間は計測できる見込みである．1. 制約条件の入力法と，3. 入力法に対応して実行時間と比較して条件判定する手段を実現すれば良い．

- デバッグ支援ツール
  DEM では，実行順序に影響を与える要素が，1. ディスパッチテーブルを介した登録と実行処理と，2. ロックを使った共有メモリへのアクセスだけである．収集したログデータから，これらの実行タイミングと実行制御情報を記録可能と判断した．また，DEM では並列実行の制御方法が明らかなため，記録したデータからの再現実行制御を実現できる見込みである．ただし，外部入力の再現とシステムコールなどの影響を再現する方法は，別途実現方法を検討する必要がある．

最後に，DEM-PPE の他システムへの適用について検討する．DEM-PPE は今回の開発対象アプリケーション向けの並列処理機構 (DEM) で実現されており，一般的な並列処理で利用することは難しいが，今回の適用先と同じ端末アプリケーションならば適用可能性は高い．また，DEM-PPE は標準的な Posix の Pthread Library をベースとしているため，システムへの依存度は低く，他のプラットホームへの移植性は高い．

6. おわりに

開発対象のアプリケーションに適した並列処理機構を実行モデルとして定め，定めた実行モデルに基づく並列プログラム開発支援環境 “M-PPE” の構築法を提案した．M-PPE をリアルタイムアプリケーションである監視・制御システムの操作端末のソフトウェアに適用し，その効果を検証した．

適用にあたっては，対象アプリケーションに適した実行モデルとしてディスパッチ実行モデル “DEM” を規定した．次に，DEM に基づいた支援環境 “DEM-PPE” として簡易ライブラリの “ディスパッチルーチン” と性能評価ツールを開発した．これらのツールを使って実際にリアルタイムアプリケーションを開発し，次の成果を得た．

- プログラム作成の負荷を低減
  並列処理機構を提供することで，開発者の作業負荷が低減することを確認した．また，実行モデルを定めることで，開発者間で並列実行制御による不整合を効率良く回避できることを確認した．
- 性能評価の効率化
  実行モデルを定めることで，有用な実行情報を効率良く自動収集できること，解析と表示ツールを容易に実現できることを確認した．また，DEM 情報の収集で外部入力とログ情報の対応関係を取るための “ディスパッチ ID 設定機能” を実現した．
- リアルタイム制約検証と再現実行の実現性を検討
  収集した実行ログ情報から，リアルタイム制約検証で判定基準となる実行時間情報の計測と，再現実行で基準となるポイントでの実行情報の収集に目処を得た．

今後の課題は以下の通りである．

- アプリケーションタイプ別の実行モデルの開発
  実行モデルに基づく並列プログラミング支援環境は，現在 DEM の他に “数値演算を対象としたループと関数の並列化[11]” のみ実現している．今後実行モデルを増やして，適用範囲を広げる．
- リアルタイム検証と実行再現機能の実現
  リアルタイム検証と実行再現機能については課題を明らかでき，実現への目処を得た．これらの機能をリアルタイム検証ツール，デバッグ支援ツールとして実現していくこと．

参考文献

1) Ziya Aral and Ilya Gertner: *Non-intrusive and interactive profiling in Parasight*, ACM SIGPLAN NOTICES, 23(9):21-30,1988.9.
2) Ziya Aral and Ilya Gertner: *High-level debugging in Parasight*, ACM SIGPLAN Notices, 24(1):151-162,1989.1.
3) SUN Soft: SPARCWORKS/iMpact lock_int ユーザガイド, 1995.1.
4) SUN Soft: SPARCWORKS/iMpact スレッドアナライザユーザガイド, 1995.1.
5) SUN Soft: SPARCWORKS/iMpact ループツールユーザガイド, 1995.1.
6) 石川裕 他: 並列プログラミング言語 MPC++ の実現,JSPP '94, 105-112, 1994.
7) David J. Jablonowski: *VASE: The Visualization and Application Steering Environment*, SUPER COMPUTING '93, 560-569, 1993.
8) T. J. LeBlanc and J. M. Mellor-Cummey: *Debugging parallel program with instant replay*, IEEE Trans. on Computer, vol.C-3, No.4, 471-482, 1987.
9) 本多 弘樹: 自動並列化コンパイラ, 情報処理, vol.34, No.9, 1150-1157,1993.
10) 和泉 秀幸, 中島 毅, 萩原 正敏: 実リアルタイムアプリケーションへの並列プログラミング支援環境の適応, 情処学会第 55 回全国大会, 2G-7, 1997.9.
11) 福地 雄史, 石塚 章子, 和泉 秀幸: マルチプロセッサ対応 UNIX 上での並列プログラム開発支援環境の開発, 情処学会第 48 回全国大会, 2G-9, 1994.3.
12) 和泉 秀幸, 福地 雄史: デッドロック解析・検知機能を持つマルチスレッド対応デバッガの開発, 情処学会第 49 回全国大会, 4U-1, 1994.9.