2008－OS－108 (10)
2008／4／23

メニーコアプロセッサ時代を拓くシステムソフトウェアへの挑戦

笹田耕一 † 三好健文 †
小林良太郎 †† 吉瀬謙二 †††

プロセッサ・アーキテクチャは数百のコアを搭載するメニーコアの時代へと向かいつつある．そこで我々はコアが豊富に存在するというメニーコアの特徴に着目し，ソフトウェアと協調して多機能メニーコアを実現する Feature-Packing（FP）と呼ぶプロセッサアーキテクチャ技術の開発を行っている．本発表では FP の簡単な紹介を行い，FP を利用するためのシステムソフトウェアについて，課題と検討中の方針について述べる．

A Challenge for Systems Software in the era of Many Core Processors

Koichi Sasada ,† Takefumi Miyoshi ,† Ryotaro Kobayashi †† and Kenji Kise †††

Many core that facilitates some hundreds cores will be realiezd in the near future. On the other hand, the hurdles processors are faced is becomming more diverse and severe. A multifunction many core would be important to realize. We fucused on the abundant number of cores that is the benefit of many core and study architectural techniques called *Feature-Packing* for implementing multifunction many core. In this report, we introduce the FP architecture and discuss issues of the systems software on the FP architecture.

1. はじめに

半導体技術の進歩は，チップ上に数個のコアを搭載するマルチコアの実現を可能とし[5),8),9),12)]，現在では，マルチコアが重要なプロセッサ・アーキテクチャの１つになっている．消費電力や配線遅延の増加[2),10)]などにより大規模な単一コアの性能向上がますます困難になっていく一方で，集積度は着実に向上し続けており，今後，プロセッサ・アーキテクチャは，数百のコアを搭載するメニーコアの時代となる[3)]．メニーコアでは，利用可能なコアが豊富に存在し，ハードウェア資源に対する制約が大幅に緩和される．このため，アーキテクチャの可能性が一気に広がる．しかしその一方で，半導体技術の進歩によるハードウェア資源の爆発的な増加，情報機器の用途と求められる機能の拡大などにより，プロセッサが直面する課題はより多様に，そして，より深刻になりつつある．

直面する課題において特に重要となる要求は高速化，省電力化，ディペンダビリティ向上，製造コスト低減の４つである．高速化は，プロセッサ・アーキテクチャにおいてもっとも基本的で重要な課題である．これまで高速化を実現するための研究が数多く行われ，プロセッサの性能は飛躍的な向上を遂げたが，情報機器は急速に進歩し続けており，今後も，さらなる高速化が求められる．省電力化は，モバイル機器のバッテリー駆動時間の延長，冷却コストの削減などのため，今や必須の課題となっている．ディペンダビリティとは，ユーザーから見た，情報システムへの安心感，信頼感，便利さを統合した概念[18)]である．ディペンダビリティの向上は，情報システムが人々の社会生活を支える基盤の１つとなっている現代において欠かせない課題である．プロセッサ設計や検証，歩留まりを考慮した製造コストの低減も忘れてはならない．

このように，マイクロプロセッサは，解決すべき課題を数多く抱えている．そのため，メニーコアの時代を迎えるにあたり，これらを解決する数多くの機能を備えたメニーコア・プロセッサの実現が解決策となる．そこで，我々は，コアが豊富に存在するというメニーコアの特徴に着目して，多機能なメニーコア・プロセッ

† 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 名古屋大学大学院工学研究科
Graduate School of Engineering, Nagoya University
††† 東京工業大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

サを実現する Feature-Packing（FP）と呼ぶプロセッサアーキテクチャ技術の枠組みを提案し，直面する課題の解決を目指す．

そこで，本報告では FP の概要を紹介し，FP が目指す目標を達成するためのシステムソフトウェアについて，その課題と現在検討中の内容について述べる．

以下，2 章では FP とその基本構想を紹介する．3 章でソフトウェアから見た FP について確認し，4 章で FP ファームウェアについて，5 章で FP 向けオペレーティングシステムについて議論する．そして，6 章で本報告をまとめる．なお，2 章，プロセッサアーキテクチャについては小林，吉瀬が主に文献 16) で発表したものの修正であり，3 章移行の議論は主に笹田，三好が担当した．

2. FP プロセッサアーキテクチャ

FP プロセッサアーキテクチャの目的は，アーキテクチャの技術とソフトウェアの技術を組み合わせて多機能なメニーコア・プロセッサを実現することにある．しかし，やみくもに多機能化すれば良いというわけではない．

まず，メニーコアの最大の利点は，消費電力と配線遅延を抑制しつつ，プロセッサ全体のスループットを向上させる点にあることを考慮する必要がある．個々の機能を提供するために，それに特化した回路を各コアに設けてしまうと，コアの規模が拡大し，メニーコアの利点が失われてしまう．

つぎに，アプリケーションの持つ性質が多様であることを考慮する，あるいは，プロセッサの性能，消費電力，ディペンダビリティの間にはトレードオフが存在することを考慮する必要がある．あらかじめプロセッサの規模や機能を固定してしまうと，付加した機能を活用できないといった問題や，効率が悪化するといった問題が生じる．

これらを解決するため，シンプルな構造を維持しつつ多数のコアを柔軟に制御する Feature-Packing プロセッサアーキテクチャを新たに提案する．そして，このプロセッサアーキテクチャをベースとして，多機能なメニーコア・プロセッサを実現する．本章では，FP の目標，および基本的な方針をプロセッサアーキテクチャの視点から紹介する．

我々は，ベースとなる FP プロセッサアーキテクチャを以下のように定める．

全体構成　多数の均一な Feature-Packing コア（FP コア）と呼ぶコアをアレイ状に配置し，メッシュ等のシンプルな接続網で接続することでチップを構成する．

FP コアの簡潔性　FP コアは，大規模なキャッシュ，複雑な分岐予測機構，大規模な投機処理機構などを排除し，シンプルな構成を維持する．これにより，メニーコアの利点（消費電力と配線遅延の抑制）をさらに高めることができ，さらに，タイル・アーキテクチャ[11] と同様，設計と検証のコストを抑えることができる．

Feature-Packing コアの多様性　動作させるソフトウェアを変更することにより FP コアの役割を動的に変更できる仕組みを備える．これにより，アプリケーションの実行だけでなく，それを支援する役割（例えばデータ供給の支援）も果たすことができるようになる．動作させるソフトウェアについては 3 章以降で議論する．

FP コアの融合性　複数の FP コアが協調動作（融合）して，より規模の大きい 1 つのブロックとして動作する仕組みを備える．この仕組みは Core Fusion[4] を拡張したものとして捉えることができる．この協調動作はハードウェアの仕組みとして備えることもできるが，ソフトウェアによって行うことも検討する．

FP コアの独立性　FP コアは個別に（あるいは幾つかの FP コアが融合したブロック毎に）動作周波数と電源電圧を動的に変更できる仕組みを備える．

FP コアの連続性　従来の RISC 形式の命令セットがもたらす利点の大部分を維持しながら，FP コアの間の通信を低いレイテンシでおこなう仕組みを取り入れる．これにより，従来の最適化コンパイラ技術を利用することが可能になる．

モニタリングと制御　FP コアは，自身の状況（処理のスループットや動作しているアプリケーションの性能特性，ハザードの発生頻度など）を計測し，これを報告する仕組みを備える．さらに，他のコアから報告された状況などに基づいて，他のコアに融合の指令や動作周波数変更の指令を与える（制御する）仕組みも備える．

FP では，全体構成，および，FP コアの簡潔性により，シンプルな構造を維持する．これにより，メニーコアの利点（消費電力と配線遅延の抑制）をさらに高めることができ，さらに，タイル・アーキテクチャ[11] と同様，設計と検証のコストを抑えることができる．また，FP コアの多様性，融合性，独立性などを利用し，多数の FP コアを柔軟に制御する．これによりアプリケーションの性質や，様々なトレードオフを考慮して，プロセッサの規模や機能を動的に決定すること

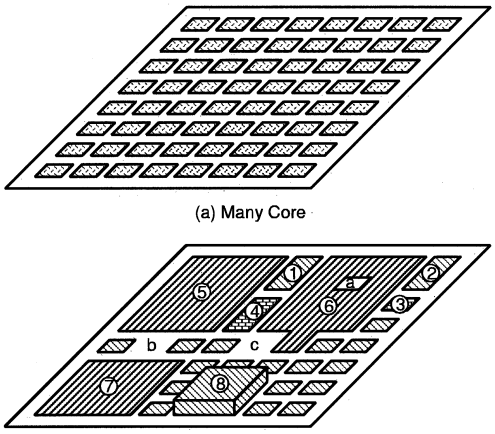

図 1 Feature-Packing many core processor

ができる．

図 1 に，Feature-Packing の概念図を示す．図 1(a) は，チップ上に均一なコアが配置された通常のメニーコアを示す．一方，図 1(b) に，Feature-Packing プロセッサの例を示す．

図において，番号が付加されていない FP コアは，単一 FP コアと同様にしてアプリケーションを実行する計算コアである．

1 番と 2 番は，計算コアを高速化するために，FP コアの融合性を利用して規模を増加させた計算ブロックを示す．

3 番は，高精度な分岐予測によって他の計算コア/ブロックの命令供給を支援するために，FP コアの多様性を利用して役割を変更した命令供給支援コアを示す．4 番は，Feature-Packing コアの多様性と融合性を利用し，3 番よりも高い予測精度を達成できるようにした命令供給支援ブロックを示す．

5 番〜7 番は，他の計算コア/ブロックに高性能なキャッシュを提供するために，FP コアの多様性と融合性を利用して，役割の変更と規模の増加を行ったデータ供給支援ブロックを示す．

8 番は，FP コアの融合性と独立性を利用して，規模と動作周波数を増加させた計算ブロックを示す．

図中の a〜c は，経年変化や製造不良といった致命的な永久故障が発生したコアや，素子ばらつきによってタイミング違反を頻繁に起こすコアを示す．こうしたコアが存在する場合，FP コアの独立性を利用して，正常なコアだけを使う．これにより，歩留まりを大幅に改善する．

表 1 従来の汎用プロセッサと FP プロセッサの違い

	従来	FP
1 チップのコア数	〜8	100〜
コアの複雑さ	複雑	単純
利用可能メモリ	大	小
コア間の通信	メモリ間	専用の手法（高速）

3. FP におけるソフトウェア

FP の目的は，前述のとおり高速化，省電力化，ディペンダビリティ向上，製造コスト低減である．FP 向けソフトウェアも，この目標を達成するものでなければならない．そこで，本章以降ではこの目標に向けたソフトウェア，とくにシステムソフトウェアの実現について考察する．

まず，ソフトウェアの視点からみると，FP プロセッサアーキテクチャは従来の汎用プロセッサと比較して **表 1** に示す違いがある．

FP はメニーコアプロセッサをベースにしているため，1 チップでのコア数がマルチコアプロセッサとして知られている現在市販されているプロセッサよりも各段に多い．コア数を確保するため，各コアはなるべく単純な構成となっている．これは，単純なコアをハードウェア・ソフトウェアにより協調動作させることで複雑な処理を行うことを想定している．同様に，コア数を多くするためにメモリについても，各コアが共有する物理メモリに直接アクセスするわけではなく，各コアが少量のローカルメモリをもつようなアーキテクチャを検討している．各コアの通信方式の詳細は検討中であるが，通信するコア間の距離による，低遅延の通信機構を備える．

このメニーコアプロセッサアーキテクチャをコア数が大きな従来通りの並列計算機として利用することも可能である．例えば N 体問題などのような大きな並列性をもち，各コアが比較的小さいメモリしか必要としないような計算を高速に実行させることは可能である．また，超並列計算機のような研究分野で議論されてきた並列プログラミングモデル，言語，そしてその処理系を FP プロセッサ用に移植することで高い性能を達成するという方針もあり得る．

しかし，本報告では，従来の汎用計算機で利用されてきた計算機環境を拡張する方針で話を進める．つまり，専用計算機とはせず，これまで利用されてきた計算モデルを拡張する形で FP の目標を達成するようなシステムソフトウェアについて考察した点を述べる．

4. FP ファームウェア

従来のプロセッサアーキテクチャでは，基本的にアプリケーションプログラマが記述した処理を各コアで動作させることが一般的であるが，マルチコアプロセッサ（もしくはマルチスレッドプロセッサ）においてアプリケーションとは直接関係のない，その実行を支援する並列実行単位を生成する研究はすでにいくつか提案されている．例えば，キャッシュヒット率を向上するためにプリフェッチを行う並列実行単位（ヘルパースレッド）で実行するような研究[6),7)]がある．しかし，コア数（マルチスレッドアーキテクチャプロセッサの場合はスレッド数）に制限があったため，多くのコア（スレッド）を積極的に利用して実行するモデルに対応したものはなかった．

メニーコアプロセッサである FP では FP コアが豊富であるため，1 つのアプリケーションが多くの FP コアを支援コアとして利用することが可能である．支援コアを含んだ FP コアの制御はハードウェアとソフトウェア，OS とユーザレベルライブラリ・アプリケーションが協調することで実現する．ただし，各 FP コアの構造を単純にするために，協調に必要となるハードウェア機構はできるだけ単純にするように検討している．なお，FP コアを制御するためのソフトウェアを FP ファームウェアと呼称する．並列度がないアプリケーションについても，様々な FP ファームウェアを搭載した支援コアを利用することでメニーコアによる利益を得ることが可能である．

FP ファームウェアによって，従来はハードウェアによって実装されていた計算機の各機構をソフトウェアによって実現することが可能になる．専用ハードウェアと比較すると，FP ファームウェアで実現した機構は単純な速度比較では不利になることが予想できるが，実行するアプリケーションに応じた適切なポリシやアルゴリズムを柔軟に取り込むことができるため，単純な専用ハードウェアに比べて有利になる可能性がある．また，実現する機構に並列度があれば，FP コアをその機構に投入することで性能をスケールすることができる．消費電力の点でいえば，FP ファームウェアの種類の特性によって動作周波数を変更することが可能である．

FP において，どの機能をソフトウェアで実現するべきか，ハードウェアで実現するべきか，ということを見極めるのは，本プロジェクトの一つの大きな課題である．また，協調に必要なハードウェアをできるだけ単純なものに抑えるのも問題である．

ここで，検討中の FP コアの種類を紹介する．

計算コア　まず，アプリケーションを実行するためのメインとなる処理を実行するため計算コアである．処理を高速化するに計算コアを増加させるには 2 つの方針がある．まず，アプリケーションがマルチスレッドに対応していた場合，単純に計算コアを増加することで性能向上を実現できる．また，内在する並列度がない処理も，Core Fusion[4)] を拡張した仕組みでより複数の計算コアを融合した 1 つの高速な計算コア（2 章で述べた計算ブロック）を構築することによって高速な実行を目指す．コアの融合方式についての詳細は，ハードウェアレベル，ソフトウェアレベルで現在検討中である．

デコードコア　プロセッサのデコードステージで行われている処理をソフトウェアで行う FP コアである．わかりやすい例でいうと，FP コア自身が解釈実行できないベクトル命令を利用したプログラムを，複数の計算コアに割り振って実行するような例が考えられる．従来のベクトル計算機とは異なり，実行時に利用可能な計算コアを考慮したデコード，ディスパッチを行うなど，柔軟な処理が可能になる．

分岐予測支援コア・データ供給支援コア　従来，ハードウェアの分岐予測器，キャッシュメモリが行っていたことを行う FP コアである．データ供給支援コアは簡単にキャッシュコアとも呼び，命令・データ用のメモリアクセスをそれぞれキャッシュする．最低限のハードウェアサポートを利用してソフトウェアで実装する[13),15),17)]．これにより，アプリケーションに関する知識に応じた，効率的なアルゴリズム・ポリシを選択可能にする．アルゴリズム以外にも，アプリケーションの特徴や規模によって利用するコア数を変更することが可能である．アルゴリズムを決定するための知識は，アプリケーション開発者によって入力する方式，言語処理系が解析した結果を利用する方式，実行時プロファイラを利用する方式によって得ることが可能である．また，リアルタイムアプリケーションのための実行時間解析を正確に行うような仕組みも検討している．

監視コア　耐障害性を高めるために，大量の FP コアについて実行時に不具合を検出する仕組みが必要がある．また，アプリケーションの実行特性を知ることができればさまざまな最適化手法を適用することができる．そこで，他の FP コアの実行をを監視する専用の FP コア（監視コア）を用意す

る．監視結果をどこに送り，どのように活用するかは現在検討中である．

MMU コア　各 FP コアは小さなローカルメモリを持っていることを想定しているが，実用上，および保護の観点からメモリ管理を行うための仕組みがあることが望ましい．そこで，メモリ管理を行うための FP コアを用意する．具体的な物理アドレス，仮想アドレスのマッピング方法は現在検討中である．

紹介した FP コアは，従来はハードウェアレベルで実装していた機構がほとんどである．ソフトウェアを利用して実装することで，どの程度の高速化・多機能化が可能になるのか，やはりハードウェアで実装しないと性能が伸びないのかは今後の評価によって明らかにしていく．

FP ファームウェアをどのように提供するのかという点にはいくつか選択肢がある．一番単純な方式はアプリケーション開発者が直接組み入れるという方式である．これを進めて，ライブラリとして提供する方式，言語処理系が提供する方式が考えられる．また，OS レベルですべての FP コアを管理し提供する方式もある．

FP ファームウェアの実行モードを，いわゆるユーザレベルで実行するのか，カーネルレベル，つまり特権をもった状態で実行するのか，それとも FP ファームウェア専用のモードが必要になるのか，現在検討中である．現在のところ，FP コアの種類によって方式が異なるだろうと予測している．たとえば，監視コアや MMU コアは OS 管理のもとで，適切な保護モードのうえで実行する必要がある．また，キャッシュコアはユーザレベルで独自に実装，もしくは言語処理系による適切なキャッシュコアの生成の可能性もあり，検討中である．ユーザレベルで管理可能とすることで，FP コアの種類の柔軟性が高まり，また OS を介在させるオーバヘッドを削減できる．

5. FP 向けオペレーティングシステム

FP コアの利用など，FP 向けのオペレーティングシステム（OS）について，従来の OS から再検討する必要がある．そこで，本稿では従来のプロセッサとは最も異なる点である FP コアの管理，そしてそれに付随するプロセス管理に絞って FP 向けの OS についての議論を行う．

なお，ここでいうプロセスとは，いわゆる UNIX でのプロセスのことであり，アプリケーションを構成する一つの枠組みを意味する．FP プロセッサアーキテクチャでは，1 つのプロセスは 1 つ以上の FP コアによって構成される．

OS に関する課題としては，他にもメモリ管理や入出力管理，割り込み処理，タイマ処理など様々な課題が残っているが，それについては機会を改めて報告する．

5.1 プロセスとコアグループ

プロセスをまたいだ FP コア間の通信を無制限に許可すると，保護の観点から問題となる．そこで，コアグループという仕組みを設けてこの問題を解決することを検討している．

コアグループはプロセスの境界を意味し，1 つ以上の FP コアの集合によって構成される．基本的に，FP コア間の通信はコアグループ内の FP コア間にハードウェアレベルで制限する．コアグループを越えた通信も必要になるが，その場合は別途通信インターフェースを設ける．

コアグループ内では，FP コアがアプリケーションの特性に応じた種類の FP ファームウェアによって動作を変更し，協調しながら実行していく．

5.2 プロセススケジューリング

FP において，OS の資源管理の一つであるプロセススケジューリングをどのように行うかは難しい課題の一つである．つまり，各プロセス（コアグループ）に対して FP コアを何個，どの位置に割り当てるかを決定する必要がある．通信時間はコアの位置から決定する距離に関する関数となるため，FP コアが 2 次元メッシュによるネットワークを構成していた場合，2 次元の配置問題になる．

なお，監視コアでの監視情報を利用して，不具合のある FP コアを利用しないようにスケジューリングする必要がある．

5.2.1 単純なコア割り当て

プロセス（コアグループ）に割り当てる FP コアの割り当てについてもっとも簡単なモデルは，M 個の FP コアがあったとき，これを静的に N 個のコアグループに分割してスケジューリングする方式である．つまり，各コアグループに M/N 個の FP コアを割り当て，OS は N 個のコアグループにプロセスを割り当てるというプロセススケジューリングを行う．

N が十分小さければ，プロセススケジューリングは従来のマルチコアプロセッサにおけるプロセススケジューリングと同等の問題になる．各プロセス内では，M/N コア内の FP ファームウェアの利用をそれぞれ行う．コアの分割を 2 次元上の固まりごとに分割すれば，コアグループ内での FP コア間の最長距離を小さくすることができる．また，コア数を必要としないプ

ロセス（アプリケーション）では，M/N より小さい数の FP コアを休止状態にすることで省電力化が可能である．

プロセスの切り替えについては，コアグループに含まれる FP コアの情報をすべて待避し，切り替え後のプロセスの情報を復帰すること実現できる．この点も，従来の OS におけるプロセススケジューリングと同等である．

このモデルは単純であり，実装も容易と予想されるが，粒度の細かいコアの割り当てが出来ないという問題がある．

5.2.2 細かい粒度でのコア割り当て

各プロセスへの FP コアの割り当てをより細かい粒度，たとえば 1 コアごとに行うためには，コアグループ同士での FP コア数の調停を行う必要がある．

必要なコア数の決定は，監視コアによるプロファイリング情報を解析し，プロセスの特性を判断することで OS，もしくはユーザレベルライブラリによって判断することができる．そして，必要に応じて FP コアの追加要求を行う．

ここで，プロセス数が増えた場合に取るべき手段が 2 つある．つまり，(a) プロセスコンテキスト切り替えを行う，(b) プロセスに属するコアグループから FP コアを減らす，という方式である．プロセス同士の公平性を保証するために (a) もしくは (b) どちらが適切かを判断する必要がある．

(b) の FP コアを減らす場合は，FP コアをどのように管理しているかによって必要な処理が異なる．FP ファームウェアの種類を OS がすべて管理している場合は OS のスケジューラ・資源管理機構による適切な増減が可能である．つまり，計算コアの増減は，従来の OS におけるカーネルレベルスレッドと同様な処理になり，キャッシュコアに代表される支援コアの増減も正しく行うことができる．ただし，この場合は FP コアの種類の増減，種類の決定をすべて OS へ依頼（システムコール）する必要があり，効率が悪い．

対して，FP コアをある程度ユーザレベルで管理していたときには，FP コアを減らすときに 2 レベルスレッド管理機構で行われてきたような OS からユーザレベルへの通知を行うなどの処置が必要になる．つまり，OS はあるプロセス（コアグループ）から FP コアの数を減らすことを決定したとき，取り除いてから，もしくは取り除く前にユーザレベルへアップコールなどを用いてその旨を通知し，ユーザレベルで適切な処置を行わせる必要がある．これは，Scheduler Activations[1),19)] のような OS とユーザレベルライブラリ，アプリケーション間での協調技術と考えることができる．ただし，従来はユーザレベルでのスケジューリング対象が計算コア相当のものだったが，この場合支援コアの存在も意識する必要があることが従来と異なる．例えば，OS によってキャッシュコアが取り除かれたとき，正しく動作を続行させるための機構は自明ではない．アプリケーションによって重要な FP コアの種類が異なる．また，適切に取り除く FP コアを決定するためにはカーネルレベル・ユーザレベル間の協調・連携を行う必要がある．

5.3 システムコールとコアグループのピンダウン

プロセスの数が増えればコアグループの待避，復帰が必要になるが，いくつかの種類のコアグループは切り替えの対象としないほうがよい場合がある．例えば，OS の諸機能を司る OS コアグループがそれにあたる．

我々はユーザレベルアプリケーションに対して OS の機能をどのように提供するか，つまり，システムコールの実現手法について次の 2 つを検討した．まずは従来の OS を踏襲する手法で，システムコールを行うと実行モードを特権モードに遷移し OS の機能を利用する方式がある．つまり，ある計算コアが特権モードになり，OS のコードを実行することになる．それに対して，OS コアグループを用意し，システムコールはそのコアグループへの通信によって実現する，という方式がある．この場合，OS の機能を利用しない FP コアの実行モードは変わらない．

前者は，OS 管理情報へアクセスするために外部メモリへアクセスするため性能が著しく損なわれる点，OS 管理情報に対する排他制御を FP プロセッサ上で行うためにはオーバヘッドが大きくなることが予想される点から，後者を採用することにした．多くのプロセスがシステムコールを頻発すると，OS コアグループへの通信がボトルネックになる可能性があるが，OS グループ側に通信専用コアを設けるなどの回避策が考えられる．なお，前節で述べたユーザレベルへのアップコールは OS コアグループからプロセスへの通信によって実現する．

この方式は OS コアグループが常に存在しなければならないため，コンテキスト切り替え対象コアに含めないようにピンダウンしなければならない．ピンダウンすることで，OS コアグループには OS 管理データを保持する専用コアを置き，OS 制御を外部メモリとの通信なしに行う，などの工夫が考えられる．

他にピンダウンするコアグループとしては，複数のプロセスが共有するデータ共有支援コアグループ（キャッシュコアグループ）が挙げられる．FP コアが

もつローカルメモリを一次キャッシュ，コアグループに存在するキャッシュコアを二次キャッシュとすると，このキャッシュコアグループは三次キャッシュと考えることができる．ソフトウェアでキャッシュアルゴリズムを記述することができるので，アプリケーション間で協調するような効率的な共有キャッシュを実現することができる．

6. まとめと今後の課題

本報告では，今後のメニーコアプロセッサ時代に対応する，高速化，省電力化，ディペンダビリティ向上，製造コスト低減を目指すプロセッサアーキテクチャであるFeature-Packing（FP）プロセッサアーキテクチャについて紹介し，FP向けシステムソフトウェアに関する課題について，現在検討中の事柄を含めて述べた．

主にOSについて述べたため，言語処理系，とくにコンパイラについての言及は行わなかったが，例えばコンパイル時にプログラムの特性の解析，並列性の抽出による自動並列化などを行っていく必要がある．

本研究は実際にプロセッサの設計とソフトウェアの設計を互いに反映させながら進めている．この作業は，従来のハードウェア，ソフトウェアの役割分担を再検討する作業にあたる．また，OS，言語処理系，ユーザレベルライブラリ・アプリケーションについても新たな切り分けが必要である．つまり，本プロジェクトはハードウェア・ソフトウェアアーキテクチャの再構成と言える．

今後はシミュレータ[14]ベースの評価を行いながら，FPの仕様について詰めていく．

FPについての検討はまだ始まったばかりである．本報告で述べた内容は，これまでのOS研究で提示された課題・提案と多くの類似点があると思われる．ただし，メニーコアプロセッサという特徴により，通信時間やローカルメモリのサイズといった性質が異なるので，その点を注意しながら適宜サーベイを行い，新規手法の提案と既存の提案手法の取捨選択を行っていきたい．

参 考 文 献

1) Anderson, T. E., Bershad, B. N., Lazowska, E. D. and Levy, H. M.: Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, *ACM Transactions on Computer Systems*, Vol. 10, No. 1, pp. 53–79 (1992).
2) Bernstein, K.: Caution Flag Out: Microarchitecture's Race for Power Performance. Keynote Presentation to MICRO-36.
3) Borkar, S., Dubey, P., Kahn, K., Kuck, D., Mulder, H., Pawlowski, S. and Rattner, J.: Platform 2015: Intel Processor and Platform Evolution for the Next Decade, *Technology@Intel Magazine* (2005).
4) Ipek, E., Kirman, M., Kirman, N. and Martinez, J. F.: Core fusion: accommodating software diversity in chip multiprocessors, *SIGARCH Comput. Archit. News*, Vol. 35, No. 2, pp. 186–197 (2007).
5) Krewell, K.: UltraSPARC IV Mirrors Predecessor, *Microprocessor Report 2003-11-10* (2003).
6) Lu, J., Das, A., Hsu, W.-C., Nguyen, K. and Abraham, S. G.: Dynamic Helper Threaded Prefetching on the Sun UltraSPARC CMP Processor, *MICRO 38: Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, Washington, DC, USA, IEEE Computer Society, pp. 93–104 (2005).
7) Luk, C.-K.: Tolerating memory latency through software-controlled pre-execution in simultaneous multithreading processors, *SIGARCH Comput. Archit. News*, Vol. 29, No. 2, pp. 40–51 (2001).
8) Naffziger, S., Stackhouse, B., Grutkowski, T., Josephson, D., Desai, J., Alon, E. and Horowitz, M.: The Implementation of a 2-core Multi-Threaded Itanium Family Processor (2005).
9) Pham, D., Asano, S., Bolliger, M., Day, M. N., Hofstee, H. P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T. and Yazawa, K.: The design and implementation of a first-generation CELL processor, pp. 184–592 Vol. 1 (2005).
10) Pollack, F. J.: New microarchitecture challenges in the coming generations of CMOS process technologies (keynote address)(abstract only), *MICRO 32: Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture*, Washington, DC, USA, IEEE Computer Society, p. 2 (1999).
11) Taylor, M. B., Lee, W., Miller, J., Wentzlaff, D., Bratt, I., Greenwald, B., Hoffmann, H., Johnson, P., Kim, J., Psota, J., Saraf, A., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S. and Agarwal, A.: Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams, *SIGARCH Comput. Archit. News*, Vol. 32, No. 2, p. 2

(2004).
12) Tendler, J. M., Dodson, J. S., J. S. Fields, J., Le, H. and Sinharoy, B.: POWER4 System Microarchitecture, *IBM J. Res. Dev.*, Vol. 46, No. 1, pp. 5–25 (2002).
13) 吉瀬謙二, 佐々木豊: Cell プロセッサの分岐ペナルティを軽減するソフトウェア分岐予測の可能性検討 (プロセッサ・アーキテクチャ(2), 「ハイパフォーマンスコンピューティングとアーキテクチャの評価」に関する北海道ワークショップ (HOKKE-2007)), 情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2007, No. 17, pp. 245–250 (20070301).
14) 佐藤真平, 藤枝直輝, 田原慎也, 吉瀬謙二: 実用的かつコードのシンプルさを追求した Cell BE の機能レベルシミュレータ SimCell の設計と実装, コンピュータシステム・シンポジウム (ComSys2007) 論文集 (2007).
15) 佐藤芳紀, 神酒勤: Cell Broadband Engine への SPE ソフトウェアデータキャッシュの実装 (最適化・高速化), 情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2007, No. 59, pp. 13–18 (20070608).
16) 小林良太郎, 吉瀬謙二: 多機能メニーコアを実現するアーキテクチャ技術 Feature-Packing の構想 (Inventive and Creative Architecture 特別セッション I), 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, Vol. 2007, No. 115, pp. 11–15 (20071121).
17) 森谷章, 藤枝直輝, 佐藤真平, 吉瀬謙二: 多機能メニーコアにおけるデータ供給を支援するキャッシュコアの提案, 情報処理学会研究報告 2007-ARC-176 (2008).
18) 入江英嗣, 荻野健, 勝沼聡, 清水一人, 栗田弘之, 五島正裕, 坂井修一: 超ディペンダブル・プロセッサアーキテクチャの構想 (ディペンダブルプロセッサ, ディペンダブルコンピュータシステムとセキュリティ技術及び一般), 電子情報通信学会技術研究報告. DC, ディペンダブルコンピューティング, Vol. 106, No. 4, pp. 49–54 (20060407).
19) 猪原茂和, 益田隆司: ユーザとカーネルの非同期的な協調機構によるスレッド切り替え動作の最適化, 情報処理学会論文誌, Vol. 36, No. 10, pp. 2498–2510 (1995).