計算システム工学分野

# メニーコアプロセッサ構成の検討を目的とした高速トレースシミュレータの開発

学籍番号 23413575 氏名　山田 龍寛

指導教員名　津邑 公暁 准教授

## 1　はじめに

今後，集積度の向上に伴って，搭載するコア数を増大させたメニーコアプロセッサが一般化すると予想されている．しかし，コア数の増大に伴い，一部のコアに処理を割り当てきれず，全てのコアを有効に利用できない状況が発生することが予想される．

そこで本研究では，安定したデータ供給が可能なプロセッサ構成の実現可能性を検証するために，配線遅延を考慮したプロセッサ構成方式を設計し，その実行トレースを採取可能なメニーコアトレースシミュレータを開発する．加えて，メニーコア研究において，シミュレーション時間の増大が深刻な問題となっている．これを解決し，大規模なアプリケーションを現実的な時間で実行するために，シミュレーション自体を高速に行う機能を開発したシミュレータに追加実装する．

## 2　マルチコア／メニーコアプロセッサ

今後は半導体のプロセスルールの縮小に伴い，単一プロセッサ当たりに搭載されるコア数がさらに増大し，メニーコアプロセッサが一般化すると予想される．マルチコア／メニーコアプロセッサは，データ供給の転送効率を向上させるキャッシュシステムと，複数のコアやメモリ間の交信路を提供する相互結合網のトポロジについて様々な構成方式が存在している．

## 3　既存シミュレータの問題点とその解決方針

これまでに数多くのマルチコアプロセッサシミュレータが開発されているが，これら既存のシミュレータはいずれも共有メモリを前提として構築されている．しかし，メニーコアプロセッサでは全コア共有のメモリシステムを想定することは現実的ではなく既存のシミュレータではシミュレーションを行うことは難しい．そこで，本研究ではメニーコアプロセッサの理想的な構成を検討するために，様々な構成方式を検証可能なメニーコアシミュレータを開発する．

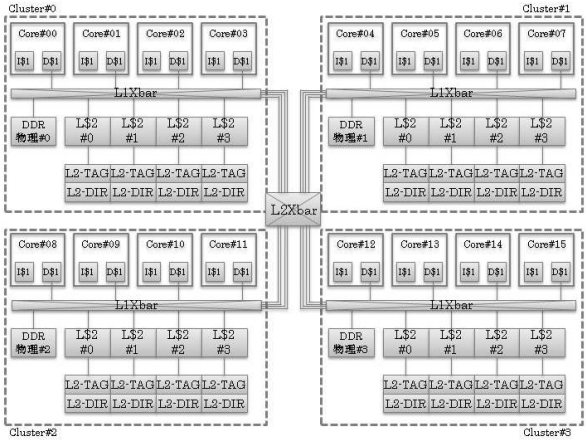


図 1 ： アーキテクチャ構成

## 4 メニーコアトレースシミュレータの開発

キャッシュ構成はプロセッサコア数を増大させることを考慮して，L2 キャッシュを複数のバンクに分割したクラスタを複数備える構成とし，ディレクトリベースの一貫性管理機構を持たせる．キャッシュコヒーレンシプロトコルは，各クラスタの持つ 2 次キャッシュに，メインメモリへのアクセス回数を減少させるために，書き戻し頻度を削減できる MOESI プロトコルを採用する．また，各コアの持つ 1 次キャッシュに，一貫性管理の複雑度を抑えつつ効率的なキャッシュアクセスが可能な MESI プロトコルを採用する．相互結合網の形状には，データ転送性能，ハードウェア物量，消費電力のバランスが最も良いと考えられる階層型クロスバ結合網を採用する．

以上で述べた構成方式をもとに設計したアーキテクチャの構成図を図 1 に示す．最小構成として 16 コア内蔵の 2 階層型クロスバ結合の cc-NUMA アーキテクチャとなっている．また，全体で 4 クラスタ構成となっており，1 クラスタに 4 つのコアが内蔵されるとする．そして，各コアと 2 次キャッシュ(L$2)がクロスバに接続され，各コアはローカルに命令キャッシュ(I$1)とデータキャッシュ(D$1)を内蔵している．また，主記憶装置 DDR はメモリの帯域幅を考慮して各クラスタに均等分割し，クロスバの 1 つに接続している．

表 1 ： 実行サイクル数

| | exec | I1wait | D1wait | cycle |
|---|---|---|---|---|
| 1thread | 7608989 | 212257 | 150306 | 7971552 |
| 2thread | 7385398 | 212240 | 95280 | 7692918 |
| 4thread | 7273194 | 212016 | 70446 | 7555656 |
| 8thread | 7219029 | 212224 | 70780 | 7502033 |
| 16thread | 7203082 | 213352 | 78876 | 7495310 |

## 5 シミュレータの動作検証

動作検証プログラムには，汎用ベンチマークプログラムである SPLASH-2 の FFT を用いた. 動作結果を表 1 に示す．各コアに 1 スレッドずつ割り当て，1, 2, 4, 8, 16 スレッド実行でそれぞれシミュレートした．なお，紙面の都合上，メインスレッドの結果のみを示している．スレッド数の増大に伴い実行サイクル数の exec が減少する結果となった. 台数効果を得られていることからシミュレータが正しく動作していることがわかる. しかし, 16 並列実行時には, 8 並列実行時に比べて D1wait のサイクル数が増大してしまっている．そのため，さらに並列度を上げた場合には総実行サイクル数が増加に転じる可能性がある．これにより，高並列に実行可能なプロセッサ構成の検討が必要であることが改めて認識できた.

## 6 シミュレーションの高速化

ソフトウェアによるアーキテクチャシミュレーションは評価に多大な時間を要し，コア数の増大に伴うシミュレーション時間の増加が深刻な問題となっている．そのため，これまでに数多くのシミュレーション高速化手法が提案されてきた. そのなかでも, シミュレーションを分散・並列化することで高速化する手法が効果を上げている[1]. そこで, 開発したシミュレータを実マルチコア環境においてスレッド並列化することでシミュレーションを高速化させる.

しかし，単純にシミュレーションをスレッド並列化するだけでは，実マシンのコア数に対して生成するスレッド数が多くなりすぎる場合があり，かえって実行時間が増加してしまう可能性がある. そこで, 実験環境に合わせてシミュレータ自身がスレッド数を調整する並列度の自動調整機能を，開発したシミュレータに追加実装した．これにより，ユーザへの負担を増やすこと無く，シミュレーションにかかる時間を削減減することができる.

## 7 シミュレーション時間の評価

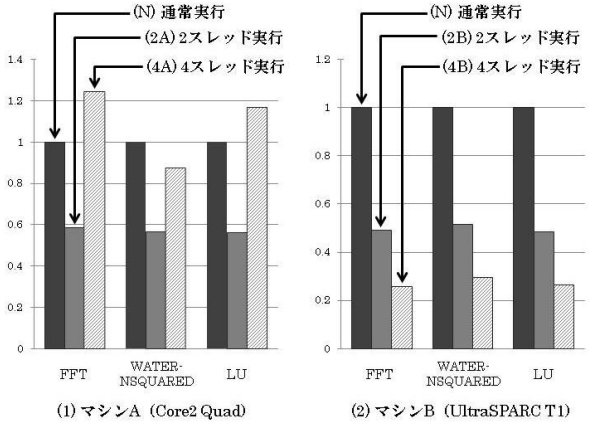


図 2 ： シミュレーション時間の評価結果

評価対象プログラムには SPLASH-2 の FFT, LU, WATER-NSQUARED を用いた. 各プログラムのシミュレーション時間を図 2 に示す．グラフは左から，通常実行した場合，2 スレッドで並列実行した場合，4 スレッドで並列実行した場合のシミュレーション時間をそれぞれ表しており，通常実行時を 1 として正規化している．図 2(1)は Core2 Quad を搭載したマシン A で実行した結果，図 2(2)は UltraSPARC T1 を搭載したマシン B で実行した結果を示している.

(4A) では物理コア数と同じスレッド数で実行したため，OS と資源の奪い合いをしてしまい，シミュレーション時間が増加してしまっている．一方で, (2A)は全プログラムのシミュレーション時間を削減できており，(N) に比べて平均で40.9%，最大43.8%のシミュレーション時間の削減に成功した.

マシン B 上は，スレッド数に関わらず全てのプログラムでシミュレーション時間の大幅な削減に成功した．UltraSPARC T1 は並列処理性能が高いため, 特に (4B) では (N) に比べて平均で 72.6%，最大で 81.8%のシミュレーション時間の削減に成功した.

## 8 おわりに

本研究では，配線遅延を考慮したプロセッサ構成を設計し，その実行トレースを採取可能なメニーコアトレースシミュレータを開発した．加えてシミュレーションを高速化させる機能を追加実装した.

## 6 参考文献

[1] Potuzak Tomas: Distributed-Parallel Road Traffic Simulator for Clusters of Multi-core Computers, Proc. 16th IEEE/ACM Int'l Symp. On Distributed simulation and Real Time Applications(DS-RT), pp.195-201(2012).