Vol.2022-ARC-248 No.19
Vol.2022-SLDM-198 No.19
Vol.2022-EMB-59 No.19
2022/3/10

次世代高性能計算ノードに向けた メモリアーキテクチャ探索のためのツールチェーン

幸 朋矢[1,a)]　遠藤 敏夫[1]

概要：半導体微細化による性能向上が物理的に難しくなるなか，近年高性能計算ノードでは異種メモリの採用やチップレットによるキャッシュの導入などによって性能向上を狙う例が散見される．今後もこのノード内アーキテクチャデザインの複雑化はさらに進むと考えられるが，そこで重要となるのが，考案したデザインが種々のワークロードにおいてどのように動作するか，どの程度の性能が得られるかを推定するツールである．本稿では，既存のサイクルアキュレートシミュレータと比較してより軽量で扱いやすいメモリシステム性能推定ツール PMNet と，幅広いアーキテクチャトポロジーを表現可能な GUI ツール DAT Viewer を中心としたツールチェーンについて報告する．

1. はじめに

近年の高性能計算機の開発において，さらなる半導体微細化が困難となっていくなか，チップレットの採用によるさらなるメニーコア化や，異種混合メモリやチップ積層などにより性能向上をねらう傾向にある．今後もさらに新しい技術の導入によりアーキテクチャのデザイン空間も広がるため，その中からより良いデザインを探し出す必要性が高まっている [1]．特にメニーコア間および複雑化するメモリ階層の接続に注目した，迅速なアーキテクチャ探索が求められている．

現状のアーキテクチャ探索においては gem5[2] や SST[3] といったアーキテクチャシミュレータが広く用いられているが，多くはサイクルアキュレート動作をベースとしており，シミュレーションに長大な時間を要することが多い (アプリ本来の実行時間の 1 万倍など)．そのため元から実行時間が長いアプリをシミュレータ上で動かすことは難しく，またマイクロベンチマークにおいても結果が得られるまでの時間が長いため，探索できるアーキテクチャの種類が限られてしまう．考案したデザインがどの程度優れているかは様々なワークロードにて評価する必要があるため，シミュレーション速度は常に重要となる．さらに，シミュレータが前述したような複雑化するアーキテクチャデザインや新技術に対して迅速に対応できるかという課題もある．各研究者は，計算ノードのモデリングを行う設定ファイルを各々のツールの混み入った仕様を理解しつつ記述する必要があるため，全く新しいアーキテクチャを表現しようとした場合の導入コストが高い．今後複雑化が予想されるアーキテクチャデザインのトポロジーをユーザーがテキストファイルで記述する負担も大きい．

我々はこれらの問題に着目し，既存のシミュレータと比較して軽量に動作する性能推定ツール PMNet を開発した．今後システム内において演算速度よりデータ移動速度によるボトルネックがより深刻になるという動向 [4] を踏まえ，このツールではメモリシステムの探索に焦点を当て，サイクルアキュレート動作の代わりにメモリトレースドリブンな動作を採用することで，既存のシミュレータでは得られなかったスピードを実現した．また，シミュレーションしたいマシンをモデリングする際，ユーザーがテキストファイルにてトポロジーを記述することは今後複雑化するアーキテクチャデザインの探索においてはコストが高いと考え，直感的にノード内インターコネクトトポロジーを構築・表現可能な GUI ツール DAT Viewer を開発した．このツールでは PMNet で使用するマシンモデリングファイルを作成する機能に加え， PMNet で得られた性能推定結果を反映させ，どの部分がボトルネックとなっているかなどの情報をグラフィカルに表示する機能も搭載した．

一般にシミュレーション速度と精度の間にはトレードオフがあり，サイクルアキュレートとしないことにより精度は下がる傾向にある．PMNet においては，計算ノード中のキャッシュやメモリなどの各構成要素のスループットと占有時間の推測に焦点をおくことで速度向上を実現したが，スループット由来以外のストール時間など，捕捉していない要素も存在する．このような性質を持つ PMNet につい

[1] 東京工業大学 学術国際情報センター
[a)] yuki.t.ab@m.titech.ac.jp

て，速度と精度の双方を評価する．

本稿では，まず既存のシミュレータについて紹介し，我々の開発したツールとの差異を明確にする．我々のツールの詳細と，ツールを使用して実際に性能推定を行う際に必要となる種々の細かいツールも含めたツールチェーン全体の解説を行う．また，シンプルなマシンを想定した際の NAS Parallel Benchmark におけるシミュレーション速度・精度両面を既存シミュレートと比較しつつ評価し，さらに 128 コアマシンにおける Triad シミュレーションの評価も行う．

2. 関連研究

本章では既存の性能測定ツール，シミュレータについて主要なものを紹介する．

2.1 Siena

Siena[5] は ORNL (Oak Ridge National Laboratory) が開発するメモリシステム探索のためのフレームワークである．Aspen[6] と呼ばれる DSL にモデリングしたいマシンのスペックと抽象的なアプリケーションを記述することで，メモリアクセスパターンを自動生成し，Ramulator, DRAMSim2，NVDIMMSim といった外部のメモリシミュレータにデータを流すことで簡易的にメモリシステムシミュレーションおよび性能推定を行う．Siena では実際のアプリケーションバイナリを用いずユーザー定義の Aspen からメモリアクセスパターンを生成するため高速に動作する一方，調査したいアプリごとにユーザーが Aspen を記述する必要がある．また，アプリやマシンの表現の幅は Aspen の仕様に制限される．例えばアプリにおいては計算と通信を交互に細かく行うようなワークロードは記述しづらく，マシンモデリングにおいては基本的な階層型のシステムしか構築できず，自由度の高いトポロジーを記述する表現力はない．また外部メモリシミュレータはサイクルアキュレート動作のため，それらを用いる場合はシミュレーション速度も律速される．

本稿で解説する PMNet の，キャッシュシミュレーション部分やオブジェクト定義部分は，この Siena を参考に開発されている．しかし PMNet はシミュレーション速度向上のためにサイクルアキュレートではなく，マシンモデリングに用いる情報やアプリケーション情報も Siena とは異なり，別のツールとなっている．

2.2 gem5

gem5[2] は様々な用途で広く使われているシステムシミュレータである (A64FX の開発のためつくられた理研シミュレータも gem5 を使って開発されている [7])．幅広く細かい設定が可能で，CPU モデルであれば軽量なものからパイプラインや Out of order の挙動を再現できるモデルまで用意されており，CPU からキャッシュ・メモリまでサイクルアキュレートでシミュレーションする．キャッシュコヒーレンシについては SLICC と呼ばれるプロトコルを用いることで独自のコヒーレンシを定義できる．システム全体をシミュレーションする FS モードと，システムコールをエミュレートして比較的軽量に動作する SE モードの 2 つが用意されている．シンプルなマシンを記述する Classic システムに加え，Ruby や Ruby Garnet といった，より幅広いトポロジーが表現できるマシンモデリング表現方法が存在する．既存のシミュレータの中では精度が高い一方，シミュレーション速度は遅く，詳細な設定を記述するコストも高い．

2.3 SST

SST (Structural Simulation Toolkit)[3] は SNL (Sandia National Laboratories) が開発するシステムシミュレータである．コアと複数のコンポーネントから成り，ユーザーは用途に応じて適宜コンポーネントを選択することができる．CPU モデルのコンポーネントとしては Ariel や Prospero といったメモリトレースベースで動作するものがメインとなるため，これらを使用する場合 gem5 と比較すると CPU エミュレートの精度は下がる．memHierarchy と呼ばれるコンポーネントにてマルチノードも含む幅広いシステム・トポロジーを表現可能である．各コンポーネントは並列に動作させることが可能で，マルチスレッド・マルチプロセスにてシミュレーションを行うことにより，大規模なシステムに対して並列化の恩恵を受けてシミュレーションを行うことができる．SST のコアはサイクルアキュレートで動作する．memHierarchy のコンフィグは gem5 の設定ファイルよりは記述が用意ではあるが，ユーザーがテキストベース (Python) で記述する必要がある．典型的なトポロジー (メッシュ等) については Merlin や Kingsley などといったコンポーネントを用いることで容易にマシンモデリングが行えるようになっている．

3. PMNet ツールチェーン

3.1 ツールチェーン全体像

図 1 にツールチェーン全体像の概要を示す．中心的なツールとして性能予測を行う PMNet には，大きく分けて対象アーキテクチャの情報 (図の Arch Info) と，アプリケーションの情報 (App Info) が与えられる．前者は DAT フォーマットと呼ばれる形式のファイルで表され，章を分けて解説する．

3.2 アプリケーション情報の取得

本ツールチェーンで性能推測の対象とするアプリケーションは 1 プロセスとし，複数スレッドを含んで良い．そのようなアプリケーションを対象に，メモリトレースデータと演算回数統計データ (optional) の情報を抽出して用い

Vol.2022-ARC-248 No.19
Vol.2022-SLDM-198 No.19
Vol.2022-EMB-59 No.19
2022/3/10

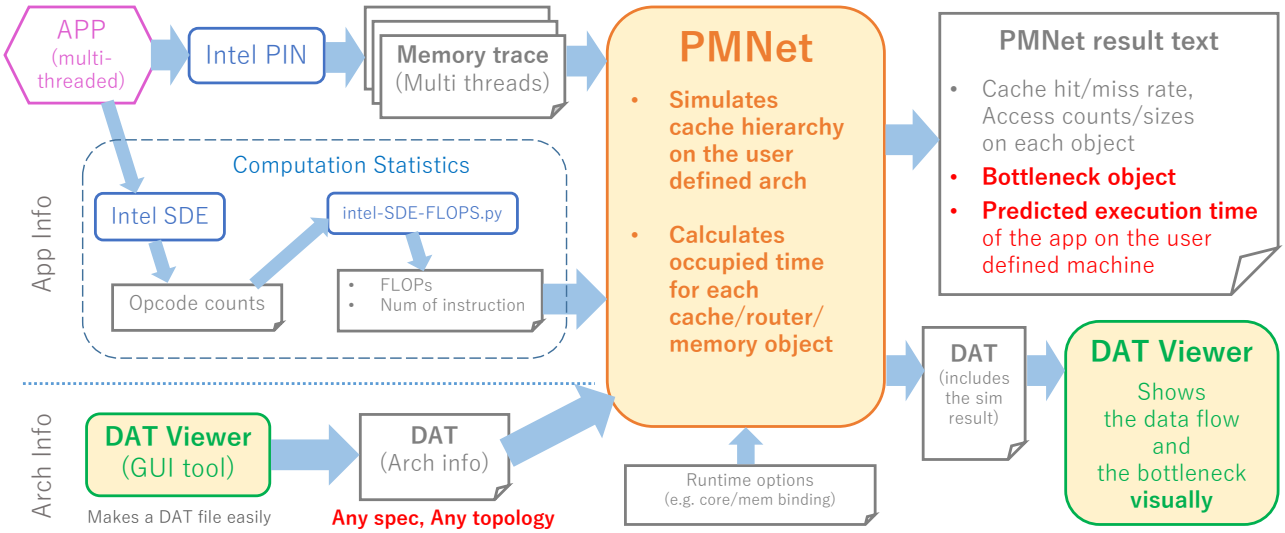


図 1: PMNet Toolchain Overview

る．アプリケーションの情報抽出は，それを既存の計算機上で後述のように動作させることにより行う．動作に用いる既存計算機は，推測対象の計算機とコア数を含め異なるアーキテクチャでよいが，現在は既存計算機は x86_64 アーキテクチャである必要がある．

本ツールチェーンでは，アプリ中の全メモリアクセスについてのメモリトレースデータを用いる．一回のアクセスについて，対象アドレス，アクセスサイズ，読み/書きの種別からなるデータを取得する．これらのメモリトレースデータについて，既存のバイナリ書き換えフレームワークである Intel PIN 上で，対象アプリを動作させてファイルに書き出す．本ツールチェーンではこのための PIN のカスタムパスを用意している．マルチスレッドのアプリを動作させた場合は，スレッドごとにトレースファイルを出力する．これらのトレースファイルのサイズは莫大となるために，入出力の時間コストやストレージ容量不足の問題が生じる．これらを軽減するために，UNIX 系 OS の Named Pipe 機能を使いメモリトレースを取得しながら PMNet で同時に読み込みシミュレーションを進めることもできる．

メモリトレースに加えオプションとして，演算回数統計データを用いることができる．これには CPU コアごとの浮動小数点演算回数 FLOPs と Instruction 数の 2 種が含まれる．これらは PMNet を動作させるにあたり必須な情報ではないが，キャッシュ・メモリにおけるデータ移動のコストに加え CPU コアにおける演算コストも加味した性能推定が可能となる．PMNet ツールチェーンでは，Intel SDE[8] でまず命令種類別の統計データを取得し，それを Intel-SDE-FLOPS[9] というスクリプトにて FLOPs 情報にまとめ上げる．本来演算回数の取得はメモリトレースを取るためのアプリ実行と同タイミング，即ち Intel PIN により同時取得することも理論上可能だと考えられるが，どの命令が浮動小数点数演算かを判別することと，マスク演算においてレジスタの中身を見ながら実際の演算回数を正しくカウントすることが実装上困難だったため [10]，現状 Intel SDE を採用している．

3.3 DAT: アーキテクチャ情報のフォーマット

カテゴリ	スペック名	説明
core_class	name	クラス名
	dp_flops	DP FLOPS
	sp_flops	SP FLOPS
	ips	IPS
cache_class	name	クラス名
	capacity	キャッシュ容量
	associativity	連想度
	linesize	キャッシュラインサイズ
	read_bandwidth	読み込みバンド幅
	write_bandwidth	書き込みバンド幅
mem_class	name	クラス名
	capacity	メモリ容量
	linesize	ラインサイズ
	read_bandwidth	読み込みバンド幅
	write_bandwidth	書き込みバンド幅
router_class	name	クラス名
	read_bandwidth	読み込みバンド幅
	write_bandwidth	書き込みバンド幅
edge_class	name	クラス名

表 1: Class list in DAT

我々は，シミュレーション対象のアーキテクチャの表現方法として，DAT (Definition of Architecture Topology) ファイルフォーマット の仕様を考案した．DAT は JSON フォーマットで記述される．コア，キャッシュ，メモリモジュール，ルーターなど (ここではオブジェクトと呼ぶ) を 1 つずつ定義していくオブジェクト群と，それらのスペック情報を記述するクラス群により構成される．**表 1** にクラ

Vol.2022-ARC-248 No.19
Vol.2022-SLDM-198 No.19
Vol.2022-EMB-59 No.19
2022/3/10

カテゴリ	スペック名	説明
core_obj	name	オブジェクト名
	class	クラス名
	num_dp_flops	DP FLOPs
	num_sp_flops	SP FLOPs
	num_inst	命令数
	time_flops	FLOPs の占有時間
	time_inst	命令の占有時間
	time	当オブジェクトの占有時間
	numa_node	NUMA Node 番号指定
cache_obj	name	オブジェクト名
	class	クラス名
	num_read	読み込みアクセス回数
	num_write	書き込みアクセス回数
	bytes_read	読み込みバイト数
	bytes_write	書き込みバイト数
	time	当オブジェクトの占有時間
	numa_node	NUMA Node 番号指定
mem_obj	(省略)	(cache_obj と同内容)
router_obj	(省略)	(cache_obj と同内容)
edge_obj	name	オブジェクト名
	class	クラス名
	source	オブジェクト ID (無指向)
	target	オブジェクト ID (無指向)

表 2: Object list in DAT

ス，**表 2** にオブジェクトの内容を示す．DAT においてスペック情報は，各要素のスループット・バンド幅に関する情報が主となっている．これは 1 章に述べたように，PMNet がスループットに注目した性能推定を行うためである．また各オブジェクトには所属する NUMA ノード番号の設定も可能である．

DAT において，アーキテクチャは上記のオブジェクト群を接続した無向グラフとして表現される．Edge オブジェクトは 2 つのオブジェクト間の接続を表現する．表 2 では source, target という名前を便宜上用いているが，内部的に 2 オブジェクト間に区別はない．また各オブジェクトの役割はグラフ中のトポロジーから決定され，たとえば L1/L2/L3 キャッシュの区別はコアからの距離によって決定される．それ以外の種類のキャッシュなどの定義をすることもできる．

各オブジェクトには PMNet にてシミュレーションを行った後，その結果情報を入れるためのプロパティも用意されている．表 2 の青字は Intel SDE により取得される情報，赤字は PMNet により埋められる情報となっている．

図 2 に他ツールと比較した際の DAT の立ち位置を示す (設定ファイルのつくり易さは 4 種類のツールで同じマシンをモデリングをした際の筆者の経験から配置)．他ツールは基本的にユーザーがテキストベースで記述する必要があるが，DAT は GUI ツールで自動生成されることを前提にしている (スペック入力はテキストベースとなる)．Siena で用いられる Aspen は記述も容易だが同時に表現力も低い．gem5 や SST の設定ファイルは非常に詳細なスペック入力

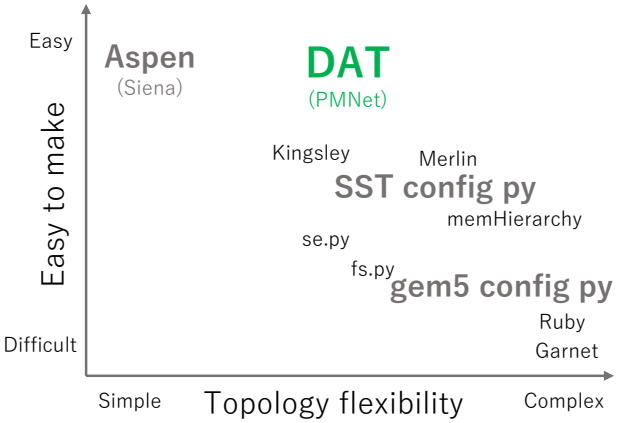


図 2: DAT の位置づけのイメージ

まで対応している一方，複雑でモデリングをしようとすると記述が煩雑になってくる．DAT はメモリシステムの評価に十分な表現力を持ちつつ，後述する GUI ツールを使うことでインスタントに生成することが可能となっている．

3.4 DAT Viewer

我々は，DAT ファイルを直感的に作成するための GUI ツールとして DAT Viewer を開発した．**図 3** が DAT Viewer のスクリーンショットとなる．右側で表示されている図は，本稿の評価でも使用する AMD EPYC gen2 Rome 2 ソケット 128 コアマシンのノード内インターコネクトトポロジーである．

本ツールは Web アプリ (クラインアントサイドの Javascript のみ) でつくられているため，プラットフォームを問わずブラウザで動作する．ネットワークグラフ表現ツールとして，Cytoscape の Javascript 版である Cytoscape.js[11] を使用している．ユーザーはマウスで直感的にコアやメモリモジュール等の各オブジェクトを配置・移動・接続することが可能となっている．この GUI で生成したマシンモデリングの情報はリアルタイムに DAT ファイルとしてテキストに反映されるため，これをそのままダウンロードし PMNet の入力ファイルとして使うことができる．

さらに，PMNet はシミュレーションの結果情報を DAT ファイルに書き込んで出力する機能があり，この結果情報入り DAT ファイルを DAT Viewer で読み込ませることにより，どのオブジェクトにどれだけのアクセスがありどれだけ時間がかかったかという情報をグラフィカルに確認することも可能となっている．

3.5 PMNet

3.5.1 PMNet の概要

これまで論じてきたツールチェーンのコアとなるのが，メモリシステム性能測定ツール PMNet (Performance Pre-

Vol.2022-ARC-248 No.19
Vol.2022-SLDM-198 No.19
Vol.2022-EMB-59 No.19
2022/3/10

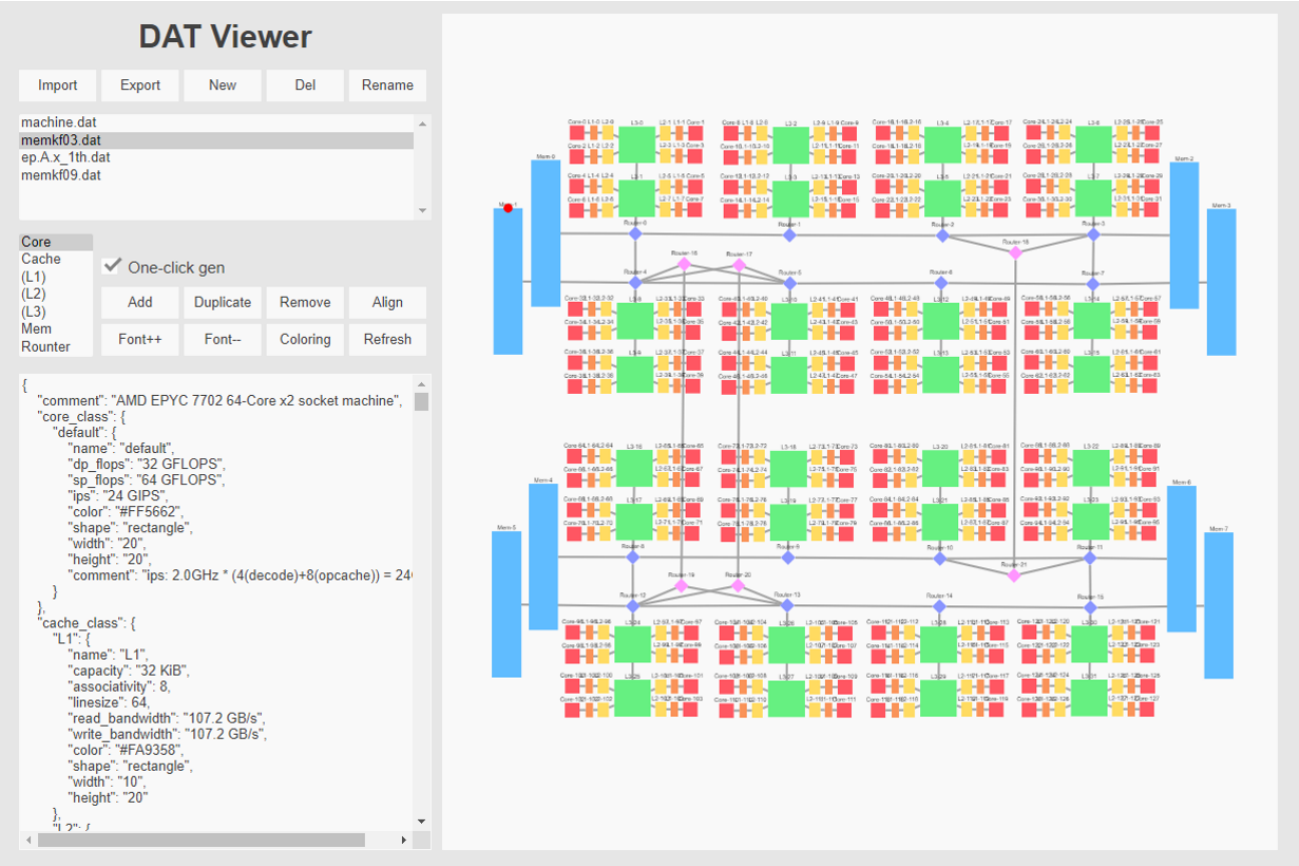


図 3: DAT Viewer Screenshot

dictor of Memory Network) である．PMNet は，これまでに説明したアプリ情報とアーキテクチャ情報を基に，推定実行時間やボトルネック箇所を出力する．

PMNet の基本的な考え方は下記の通りである：対象アーキテクチャ上で，アプリ情報である大量のメモリトレース情報を基にメモリアクセスを実行した場合，アーキテクチャ中の各オブジェクトに何回アクセスが起こったか，に注目する．そしてトレース処理の終了後，各オブジェクトの占有時間をアクセス回数とバンド幅から求め，全オブジェクト中で最長の占有時間を，アプリ実行時間の近似値として出力する．このように，アプリ実行中にはいずれかのオブジェクトがボトルネックとなっているという仮定のもと，サイクルアキュレートシミュレータなどよりも軽量に実行時間を推定する．

3.5.2 PMNet の動作

PMNet は下記のように動作する．まず DAT ファイルを基に，コア・キャッシュ・メモリ・ルータ等のオブジェクト群からなる無向グラフ構造を作成する．次に全コアから全メモリオブジェクトについての経路を，最短経路アルゴリズムを解くことにより求める．現在の実装では，ホップ数に基づいた単純なアルゴリズムを用いている．この経路はメモリアクセスがどのように伝達するかに相当し，たとえば DAT ファイルが一般的な 1 ソケットのマルチコアアーキテクチャを表すのであれば，その経路はコア・L1・L2・L3・メモリとなる．

次に，メモリトレースファイルの個数（アプリのスレッド数）分のスレッド（以下，シミュレーションスレッド）を立ち上げる．つまり PMNet 自体もマルチスレッドプログラムであり，マルチコアアーキテクチャ上のマルチスレッドアプリのシミュレーションを高速に行うことができる．各シミュレーションスレッドはまず，担当するアプリスレッドがどのコアオブジェクト上で動作するか決定する (デフォルトでは round-robin 方式で決めるが，ユーザがスレッドとコア名を明示的に対応させることもできる)．そして各シミュレーションスレッドはメモリトレースファイルを次々に読み込み，トレース 1 個あたり以下のような処理を行う．

(1) アクセス対象アドレスを基に，そのアドレスを含む OS レベルのページがどのメモリオブジェクトに存在するか表引きする．初めてアクセスされるページであれば，通常の Linux でのルールのように first-touch 方式で決定する．つまり，自コアから最も近いメモリオブジェクトを含む NUMA ノードへ割り当てる (この方式もカスタマイズ可能である)．

(2) アクセス情報の構造体を作成し，事前に求めた経路において，自コアの次のオブジェクトへ受け渡す．

Vol.2022-ARC-248 No.19
Vol.2022-SLDM-198 No.19
Vol.2022-EMB-59 No.19
2022/3/10

(3) アクセス情報を受け取ったオブジェクトの種類によって，以下の処理を行う：

- キャッシュオブジェクトが受け取った場合，キャッシュオブジェクトは内部でセットアソシアティブキャッシュのシミュレーションを行う．キャッシュヒットの場合は，現在のアクセスの処理は終了する．ミスの場合は，アクセス情報を経路上のさらに次のオブジェクトへ受け渡す．
- ルータオブジェクトが受け取った場合は，単純にそれを経路上の次のオブジェクトへ受け渡す．
- メモリオブジェクトが受け取った場合は，現在のアクセスの処理は終了する．

以上のような過程において，各オブジェクトへのアクセス回数を Read・Write ごとに記録する．また複数シミュレーションスレッドが同一オブジェクトへアクセスしうるために，各オブジェクトにて排他制御を行う．

全スレッドによる全トレースデータの処理が終了した後，アーキテクチャ内の各オブジェクトに記録されたアクセス回数，キャッシュラインサイズ，バンド幅を基に，各オブジェクトの推定占有時間を記録する．また，アプリ入力として浮動小数演算回数のデータが存在する場合は，コアの FLOPS 値で除算することにより，コアの推定占有時間を求め，記録する (命令数と IPS 値についても同様)．PMNet は，全オブジェクト間の推定占有時間のうち最長のものを，アプリの推定実行時間として出力する．

また PMNet は，各オブジェクトの推定占有時間を含めた DAT ファイルを出力する機能を持つ．これを DATViewer で表示させることにより，アーキテクチャ中のボトルネック箇所を視覚的に把握することができる．

以下に PMNet の動作についてより詳細な事項を述べる．

- キャッシュへのデータ書き込み時の動作として，あるデータをキャッシュに書き込む際に事前にキャッシュラインのデータを持ってくるという RFO (Read for ownership) の挙動をエミュレーションしている．
- 上記のシミュレーションスレッドの処理の説明において，キャッシュコヒーレンシに関する動作を欠いていた．このままではスレッド間共有データへの書き込みが無視できないアプリにおいて精度の問題があるため，単純な MSI プロトコルを実装している．このコヒーレンシ動作は write-invalidate 処理などのためにシミュレーション速度に影響があるため，この動作は ON/OFF 可能である．

3.5.3 PMNet の位置づけと課題

図 4 に，他ツールと比較した際の PMNet の立ち位置を示す (各ツールの速度と精度の関係性に関しては，各種ドキュメントや論文 [12] などを参考にした)．PMNet はメモリトレースベースで動作するため，メモリアクセスパターンを自動生成する Siena と比べシミュレーション速度は遅くなるが精度は高い．一方，gem5 や SST といったサイクルアキュレートシミュレータに対しては速度の面で優位となるも，精度の面ではアプリによっては低くなる．

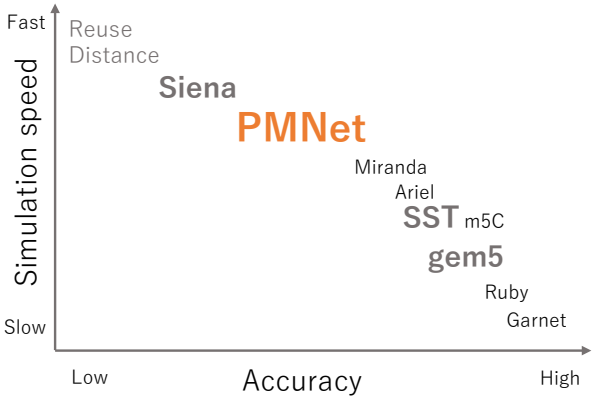


図 4: PMNet の位置づけのイメージ

現在の PMNet の精度に関する課題を，以下にまとめる．

- PMNet は実行時間の推定を，各オブジェクトにおけるスループット・バンド幅の観点から行うツールとなる．この考え方は，アプリ実行中のメモリアクセス量と演算回数のみから性能を推定する，ルーフラインモデルの一般化という側面も持つ．しかしながら，サイクルアキュレートシミュレータで用いられる，データの依存関係などの情報についてはメモリトレースファイルの時点で失われており，そこから発生するレイテンシを無視している．
- 複数のシミュレーションスレッドは (各オブジェクトにおける排他制御を除いて) 独立に走行しており，アプリ実行時の時刻やスレッド間の同期に関する情報を用いていない．この点が精度を下げる一因となりうるため，メモリトレースに時刻を追加し，緩い同期を用いることにより，シミュレーション速度と精度の両立を計画している．
- 現在はアプリ全体の処理が終了した後に占有時間を求めているため，複数フェーズから成るアプリの挙動を精度よく再現できない．これについても上述の時刻情報により改善を行う予定である．
- コアがキャッシュに書き込みアクセスを行った際，現在はキャッシュライン全体を一度下層のキャッシュからフェッチしている．しかし non-temporal アクセス命令は異なる挙動が発生する [13]．

4. 評価

この章では，まず評価に用いた環境を紹介し，次にツールの評価，バンド幅の評価に進む．

表 3 に今回の評価に用いた環境を示す．これらの環境は，シミュレーションを行うホストとしてのマシンと，シミュレーションのモデリング対象とするマシンという，2

Vol.2022-ARC-248 No.19
Vol.2022-SLDM-198 No.19
Vol.2022-EMB-59 No.19
2022/3/10

env1	CPU	Intel Xeon E5-2680 v4
TSUBAME	Node spec	14 cores 256GiB mem
3.0	Res type	q_core 4 cores 30GB mem
env2	CPU	Intel Core i7-7700K
Simple	Core	4
	Mem B/W	38.4 GB/s
env3	CPU	AMD EPYC 7702 (gen2 Rome)
Rome	Core	64 x2 = 128
	Mem B/W	51.2 GB/s x8 = 409.6 GB/s
env4	CPU	AMD EPYC 7713 (gen3 Milan)
Milan	Core	64 x2 = 128
	Mem B/W	51.2 GB/s x8 = 409.6 GB/s

表 3: Exp environment

つの意味が含まれる．環境をどちらの意味で使っているかは実験ごとに言及するため注意されたい．

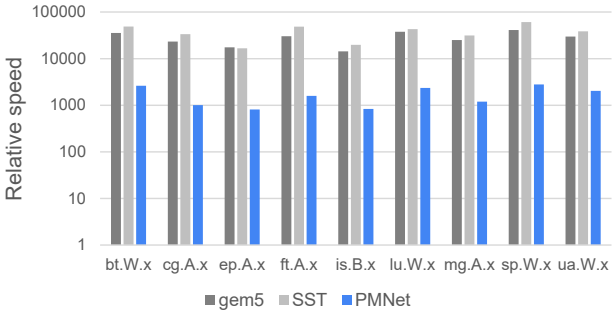
4.1 ツール間の比較

ここではPMNet と gem5 と SST を対象に，NAS Parallel Benchmark[14] にて評価を行った．ベンチマークの問題サイズは，24 時間以内にシミュレーションが終わった最大のサイズを採用している．評価環境としては env1 をシミュレーションのホスト環境とし，env2 の環境をモデリング対象とした．gem5 のシミュレーションには予め用意されている se.py という簡易モデリングスクリプトを使用した．この際マルチ L2 キャッシュ及び L3 キャッシュがこのスクリプトにて表現できなかったため，gem5 の本体及びコンフィグスクリプトにこれらに対応する修正を加えた．gem5 と SST の設定としてはシミュレーション速度が最も速い (=精度が低い) ものを選択して構成した．

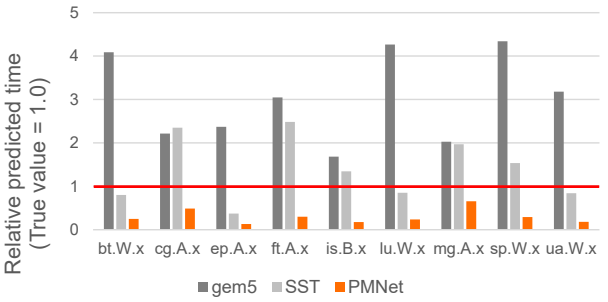
図 5 にベンチの 4 スレッド動作をシミュレーションした結果を，**図 6** に 1 スレッドの結果を示す．1 スレッドにおいては gem5 が動作しなかったため PMNet と SST のみの比較となる．シミュレーションコストは元のベンチマークプログラムの実行時間に対しシミュレーション時間がどれだけかかったかの倍率を表し，精度は元の実行時間に対してシミュレーションによる予測時間がどの程度の倍率であったかを示している．精度が 1 の場合，元の実行時間と予測時間が完全に一致していることを表す．全体の平均速度としては，PMNet が 1693 倍，gem5 が 28294 倍，sst が 37942 倍，平均精度としては pmnet が 30%，gem5 が 302%，sst が 139%となっている．

図 5(a) 図 6(a) から，gem5 と SST は元々のバイナリ実行に対し 1 万倍程度の時間がかかるのに対し PMNet は千倍程度と，大幅にシミュレーション時間を短縮できたことが確認できる．一方，**図 5(b) 図 6(b)** を見ると，精度としては PMNet が速い方向で，gem5 と SST が遅い方向で大きく外していることが分かる．gem5 と SST に関しては，今回速度比較を行うためにそれぞれのシミュレーションが最速となるように設定したため，精度を大きく犠牲にしていることに注意したい．よりマシンモデリングを細かくつくり込み，時間のかかるシミュレーションオプションを使うことで精度は上がると思われるが，同時にシミュレーション時間も増えることとなる．

PMNet は基本的にモデリングしたマシンの最大の性能を想定した推定を行うため，実際の実行時間よりも速い予測値を出していること自体は自然と考えることができる．

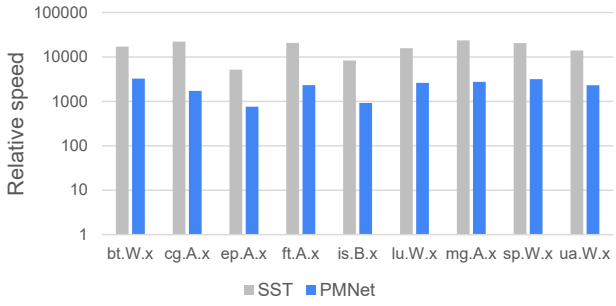


(a) Simulation cost

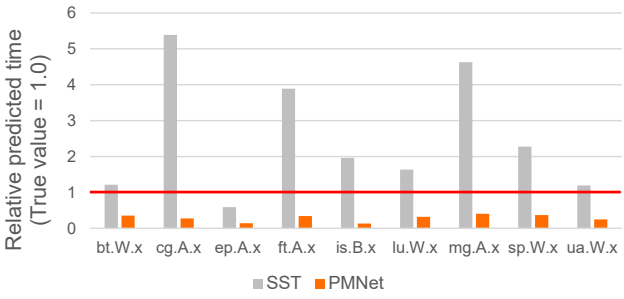


(b) Accuracy

図 5: NAS Parallel Benchmark 4 threads



(a) Simulation cost



(b) Accuracy

図 6: NAS Parallel Benchmark 1 thread

env	Conditions	B/W [GB/s] Theoretical	B/W [GB/s] Practical	Sim Speed No CC	Sim Speed MSI CC	Accuracy No CC	Accuracy MSI CC	Bottleneck
env2	4 threads	38.4	30.4	x155	x361	95%	95%	Mem
env3	16 thd NPS4	51.2	38.6	x501	x887	85%	85%	Mem
env3	128 thd NPS4 AOCC	409.6	339.3	x620	x953	79%	79%	Mem
env3	128 thd NPS4 GCC	409.6	303.3	x620	x953	89%	89%	Mem
env4	128 thd NPS4 GCC	409.6	357.5	x699	x1011	82%	82%	Mem
env4	128 thd NPS0 GCC	409.6	271.7	x975	x960	63%	63%	Router

表 4: Triad カーネルにおける PMNet のバンド幅の理論値と実効値，PMNet の速度及び精度

一方，その精度が真値である 1 から離れている理由としては複数考えられる．レイテンシなどメモリバンド幅以外の部分でボトルネックが発生しているケースや，コア内において ALU を最大限活用できていないケースなどが挙げられる．また，今回の実験ではバンド幅を理論値として DAT に記入していたが，実際の実効値としてのバンド幅は理論値よりも 10〜20%程度低くなる傾向にあることが，我々の知見から得られている．そのため，キャッシュやメモリのバンド幅の記入をより正確にすることでより真値に近くなるものと考えられる．

4.2 バンド幅評価

表 4 に，STREAM Benchmark[15] Triad を対象として複数の条件で評価した結果を示す．この実験ではシミュレーションホストとモデリング対象を同じ環境としている．つまり，env2 の実験では env2 のマシン上で env2 のマシンをモデリングをしたシミュレーションを実行している．精度の項は，今回は実効値を基準として予測性能を計算した結果で評価している．NPS (Nodes Per Socket) では EPYC プロセッサにおいて 1 ソケット内に NUMA Node をいくつ作成するかをユーザーが選択できる．NPS4 モードであれば 1 プロセッサあたり 4 NUMA Nodes 作成するため，2 ソケットでは 8 NUMA Nodes となる．NPS0 モードでは 2 ソケットを全てまとめて 1 NUMA Node とする．このモードの違いによりアプリを実行した時のデータの流れに差が生まれることに留意したい．CPU コアがメモリを確保する際，NPS4 モードでは同一 NUMA Node 内のメモリモジュール (=近いメモリ) から確保するように動くが，NPS0 では 2 ソケット全てのメモリモジュールでインターリーブする動作となる．

評価結果を見ると, DAT に実効値を記入した際の PMNet は，メモリバンド幅がボトルネックとなるベンチであればメニーコア環境であっても 80%〜95%の高い精度を示すことが分かる．一方それでも 100%にならずまだ速い予測値を出しており，この部分については今後調査し改善していく必要がある．今回キャッシュコヒーレンシ機能をオンにした場合とオフにした場合両面で比較したが，Triad の性質上，コヒーレンシによる差は精度面では見られなかった．一方でシミュレーション速度としては 1.5 倍〜2 倍程度遅くなるという結果が得られた．

最後の行の NPS0 の結果に着目すると，PMNet が推定したボトルネック箇所はルーターとなっている．これを DAT Viewer で確認すると，トポロジー的にソケットを跨ぐ位置に近接するルーターであった．NUMA Node の設定変更でデータの流れが変わり，アプリ実行においてボトルネック箇所が変わるといたケースにも PMNet は対応できるということが確認できた．一方で，その精度は 63%と低い水準に留まっている．これは，実効値を測ることができるメモリと異なりキャッシュやルーターはバンド幅として理論値を記入しており，そこに実効値との差が存在しているためという考察が可能ではあるが，今後調査が必要である．

5. おわりに

本稿ではトレースベースで動作するメモリシステム性能推定ツール PMNet と，アーキテクチャトポロジー構築 GUI ツール DAT Viewer から成るツールチェーンを紹介した．既存シミュレータとの比較を行い，PMNet ではシミュレーション速度において大きく優位性を示すも，幅広いカーネルにおける精度の課題も同時に確認した．メモリバンド幅に特化した評価では高い精度を確認し，メモリシステムの探索という観点から見た際の本ツールチェーンの有効性を示した．

本研究の今後の課題・方向性を以下に列挙する．

- さらに広いベンチマーク・カーネルでの PMNet の評価
- PMNet の精度向上
- より複雑なトポロジーでのツールチェーンの評価
- 未知メモリシステムデザインの探索

PMNet は動作をトレースベースとしたことでシミュレーション速度の高速化を得ると同時に，レイテンシ情報など種々の情報の欠落による精度の低下が課題となっている．現状の速度を維持したまま性能推定に役立つ情報を追加し幅広いカーネルで精度を上げることを狙う一方，バンド幅が効くアプリにフォーカスして，今後 CPU やメモリのスペックや構成がどのように進化するのかの予想 [16] を踏まえ，将来のノード内インターコネクトデザインや各種スペックの探索を本ツールチェーン用いて進めていきたいと

考えている．

謝辞 本稿は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP16007）の結果得られたものである．

参考文献

[1] Bharadwaj, S., Yin, J., Beckmann, B. and Krishna, T.: Kite: A Family of Heterogeneous Interposer Topologies Enabled via Accurate Interconnect Modeling, *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6 (online), DOI: 10.1109/DAC18072.2020.9218539 (2020).

[2] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Comput. Archit. News*, Vol. 39, No. 2, p. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

[3] Rodrigues, A. F., Hemmert, K. S., Barrett, B. W., Kersey, C., Oldfield, R., Weston, M., Risen, R., Cook, J., Rosenfeld, P., Cooper-Balis, E. and Jacob, B.: The Structural Simulation Toolkit, *SIGMETRICS Perform. Eval. Rev.*, Vol. 38, No. 4, p. 37–42 (online), DOI: 10.1145/1964218.1964225 (2011).

[4] Matsuoka, S., Amano, H., Nakajima, K., Inoue, K., Kudoh, T., Maruyama, N., Taura, K., Iwashita, T., Katagiri, T., Hanawa, T. and Endo, T.: From FLOPS to BYTES: Disruptive Change in High-Performance Computing towards the Post-Moore Era, *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16 (2016).

[5] Peng, I. B. and Vetter, J. S.: Siena: Exploring the Design Space of Heterogeneous Memory Systems, *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 427–440 (2018).

[6] Spafford, K. L. and Vetter, J. S.: Aspen: A domain specific language for performance modeling, *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–11 (online), DOI: 10.1109/SC.2012.20 (2012).

[7] Kodama, Y., Odajima, T., Asato, A. and Sato, M.: Accuracy Improvement of Memory System Simulation for Modern Shared Memory Processor, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPCAsia2020, p. 142–149 (online), DOI: 10.1145/3368474.3368483 (2020).

[8] Intel: Software Development Emulator, `https://www.intel.com/content/www/us/en/developer/articles/tool/software-development-emulator.html`.

[9] IT4Innovations: Intel-SDE-FLOPS, `https://github.com/It4innovations/Intel-SDE-FLOPS`.

[10] Hammond, S. D.: Towards Accurate Application Characterization for Exascale (APEX), (online), DOI: 10.2172/1221578 (2015).

[11] Franz, M., Lopes, C. T., Huck, G., Dong, Y., Sumer, O. and Bader, G. D.: Cytoscape. js: a graph theory library for visualisation and analysis, *Bioinformatics*, Vol. 32, No. 2, pp. 309–311 (2016).

[12] Butko, A., Garibotti, R., Ost, L. and Sassatelli, G.: Accuracy evaluation of GEM5 simulator system, *7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–7 (online), DOI: 10.1109/ReCoSoC.2012.6322869 (2012).

[13] Intel: Intel 64 and IA-32 architectures optimization reference manual, `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`.

[14] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V. and Weeratunga, S. K.: The NAS parallel benchmarks summary and preliminary results, *Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, pp. 158–165 (online), DOI: 10.1145/125826.125925 (1991).

[15] McCalpin, J.: Memory bandwidth and machine balance in high performance computers, *IEEE Technical Committee on Computer Architecture Newsletter*, pp. 19–25 (1995).

[16] IRDS: IRDS 2021 Systems and Architectures, `https://irds.ieee.org/` (2021).