2005－HPC－103（4）
2005／8／3

共有メモリ型アーキテクチャでの C-Stab アルゴリズムの性能評価

今　村　俊　幸†

マイクロプロセッサの複雑化に対応するために, 自動チューニング技術が提案されている. しかし, チューニングに必要なサンプリングデータの信頼性は高いとはいえない. キャッシュ競合に起因する決定的な不安定性を取り除く C-Stab アルゴリズムが提案されているが, これまでその定量的な (不) 安定性を実際のキャッシュミスの観点から論じてこなかった. 本報告では, ハードウェアモニタリングシステム (PAPI) を用いて, ベンダ実装の BLAS などでどの程度キャッシュミスが発生しているのか, また, 共有メモリ型のアーキテクチャ(具体的にはマルチプロセッサ, マルチコア) などでの定量的な評価などを実施する. そして, それらの評価を通じて性能安定化の重要性を確認することを目標とする.

Evaluation of the C-Stab algorithm on shared memory type architectures

Toshiyuki Imamura†

To apply complexifying the structure of modern microprocessors, the auto-tuning techniques are introduced, however, it is hard to trust sampled data for the tuning process in some reasons. The author is investigating the C-Stab algorithm, which removes deterministic instability caused by the cache conflict. Though mainly qualitative or consequential issues were discussed so far, no quantitative discussion was done. In this paper, we measure real cache miss rate of vendor-tuned BLAS libraries by using the hardware monitoring system, PAPI, and we evaluate the C-Stab algorithm on some microprocessor systems, specifically on multi processor and multi-core systems. We would like to aim to examine significance of the performance stability via this study.

1. はじめに

数値計算ライブラリはそのソフトウェアもしくはインタフェイスとしての寿命が極めて長いものである. 例えば, 行列基本演算ライブラリ BLAS[1] はその起源から現在に至るまで幅広いアプリケーション、特に、数値計算ライブラリの根幹部分として利用されている. その歴史は四半世紀を超えるものとなっている. この様な長い歴史の中で, BLAS に代表される基幹ソフトウェアはインタフェイスを変えずにコードそのものを変化させ今も利用され続けているのである.

基幹ソフトウェとしての数値計算ライブラリには, 利用者からの様々な性能要求に応えなくてはならない部分がある. 特に考慮すべき項目は「可般性」, 「速度性能」, 「精度」などであるが, 「速度性能」については開発者が最も心血を注ぐ部分である (Intel MKL[2], AMD ACML[3] など各種ベンダ作成の BLAS や ATLAS[4], goto BLAS[5] などを参照せよ). しかしながら, 近年のハードウェアの進展とりわけマイクロプロセッサの複雑化に伴いその最適化の実装はより困難なものになってきている. そのために, コード内に予め複数のパラメータを組み込み, 実装時または実行時に適切なパラメータを選択することで最適化を行う自動チューニング技術が一般化している. PHiPAC[6], ATLAS[4], I-Lib[7], FIBRE[8] などの自動チューニングプロジェクトの活動が盛んである.

これらの自動チューニングプロジェクトで開発されているライブラリは最適なパラメータを選択するために, 実測に基づくサンプリングが重要になるが, そのサンプリングデータの信頼性は必ずしも高いものとはいえない. なぜならば, 任意のデータレイアウトに対してその実測を包含するものとしてデータを測定しているわけではなく, あくまでも典型的なレイアウト 1 点に対して実施されたものと考えられるからである (例えば, 複数配列データの仮想記憶上の位置関係を考慮したという報告はこれまでにない). つまり, データの安定性もしくはサブルーチンの安定化を保証することなく最適化が行われているか, または, 不安定性が生じない程度に最適化を抑制しているかになる.

† 電気通信大学電気通信学部情報工学科
Department of Computer Science, The University of Electro-Communications

我々はこれまでにこの様な自動チューニングにおけるサンプリングデータの信頼性を高め, より積極的な最適化を促進させるために, 数値計算ライブラリなどに特有のデータアクセスパターンに起因するキャッシュ不安定性 (キャッシュスラッシング) を排除するキャッシュ安定化アルゴリズム C-Stab を提案してきた[11),12)]. これまでの実測に基づく実験では C-Stab アルゴリズムによってキャッシュスラッシングの擾乱は 5%以下に抑えられ, 予備的実験から十分な性能安定化がなされることが分かっている. また, 一方でそれらの測定データは実行時間ベースのものであったために, 実際にキャッシュをどの程度安定化させたかの定量的なデータがまだ示されていなかった. さらに, 従来の測定は, 単一コアのシングルプロセッサに対してのみ行われてきたために, 2005 年に一般的な PC にまで拡大された dual core CPU(または一般的な共有メモリ型のマルチ CPU) に対して, その有効性の実証が大きな課題であった.

本報告では, 現在 Intel 系のサーバとして一般的な Xeon の dual CPU と 2005 年度に新たに出現した AMD の Athlon64x2 の dual core システムを用いて C-Stab の安定性の実証を行う. さらに, C-Stab の有効性を実証と並行して, 各種ハードウェアシステムにおいてベンダーが開発している BLAS について CPU が提供するハードウェアモニタリング機能を利用してキャッシュなどの安定性がどの程度であるかの実験も行う.

2. BLAS 実装系の安定性について

図 1, 2 は, 現在 Intel 系の CPU では最速の性能を発揮するといわれている Intel Math Kernel Library(MKL version 7.1)[2)] を Intel Xeon processor(3.4GHz, Nocona コア, EM64T) 上で動作させ, その性能を測定しグラフにプロットしたものである.

このデータの興味深い点は周期的 (周期 128 次元と考えられる) な性能の落ち込みが観測されていることである. この落ち込みについては, `dgemm` の様に処理速度が大きい場合にはその時間に関する変動幅が相対的に小さい (約 10%) ためルーチン単一の評価ではそれほど目立つものではない. しかしながら, `dgemm` が計算のコア部分となる場合には, 10% の性能の揺れは大きくと考えられることが多い. 一方, 異なるコアの Xeon 2.8GHz(Prestonia コア, EM64T ではない) ではここまでの短周期の性能の落ち込みは見られず, せいぜい 4096 次元周期である. この様に, 現在最速を誇る MKL の `dgemm` でも実装系の違いや動作させるプロセッサでその振る舞いが大きく異なることが判る. また, 幾つかのルーチン (今回の測定では `dsymv`) で, 2 スレッド指定時の動作で並列処理がなされていないことも判明した.

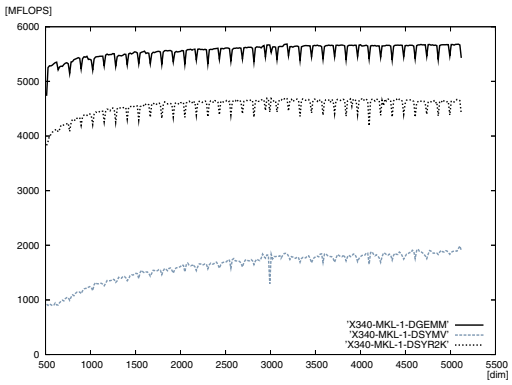

図 1 Xeon 3.4GMHz(1CPU) での Intel MKL version 7.1 の性能評価 (評価関数: `dgemm`, `dsymv`, `dsyr2k`).

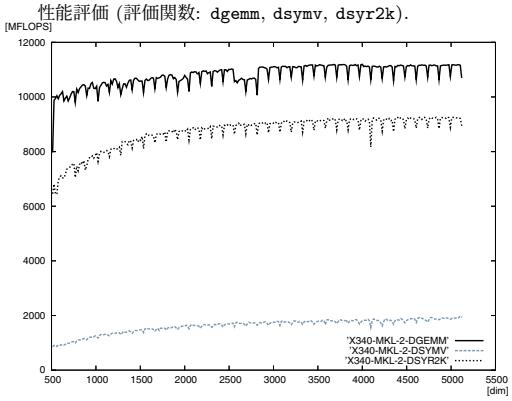

図 2 Xeon 3.4GHz(2CPUs) での Intel MKL version 7.1 の性能評価 (評価関数: `dgemm`, `dsymv`, `dsyr2k`).

2.1 ハードウェアモニタ機能による状態測定

テネシー大学によって開発されている Performance Application Programming Interface (PAPI version 3.0.8.1)[10)] (Linux カーネルでは Pettersson 氏が公開する perfctr[9)] パッチが同時に必要) は, CPU が持つハードウェアカウンタを直接制御して統一的 API により CPU の状態・動作履歴を測定するシステムである.

本システムを利用して, キャッシュに関する不安定を予見する項目の測定を実施した. 測定システムと測定項目の詳細は以下のとおりである.

[測定対象システムの詳細]

- システム : Dell 社 PowerEdge 1800, Xeon dual CPUs(3.4GHz)
- 主記憶 : PC3200 DDR2 ECC(8GB)
- L1D キャッシュ : 16KB 8way
- L2 キャッシュ : 1MB, 8way

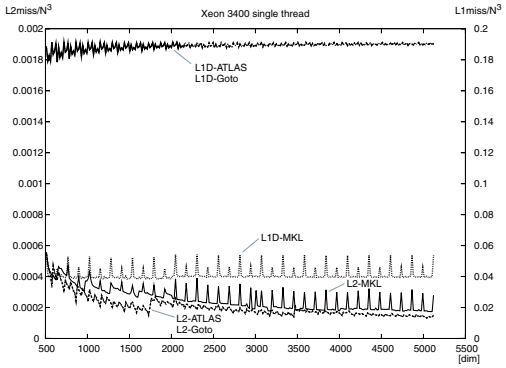

図 3 Xeon 3.4GMHz(1CPU) での dgemm のキャッシュミス率, MKL, ATLAS, Goto-BLAS の 3 種にて測定.

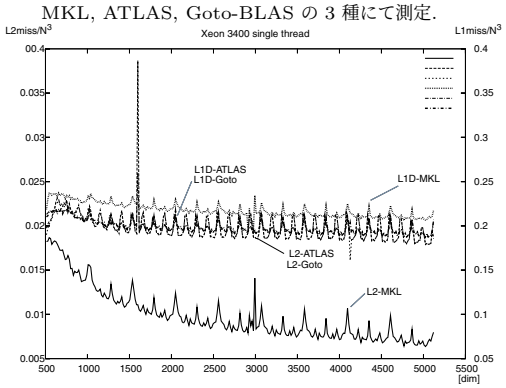

図 4 Xeon 3.4GMHz(1CPU) での dsymv のキャッシュミス率, MKL, ATLAS, Goto-BLAS の 3 種にて測定.

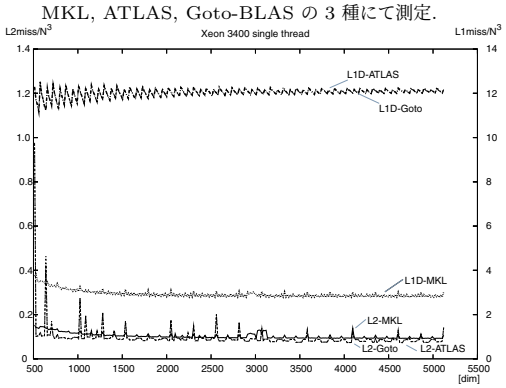

図 5 Xeon 3.4GMHz(1CPU) での dsyr2k のキャッシュミス率, MKL, ATLAS, Goto-BLAS の 3 種にて測定.

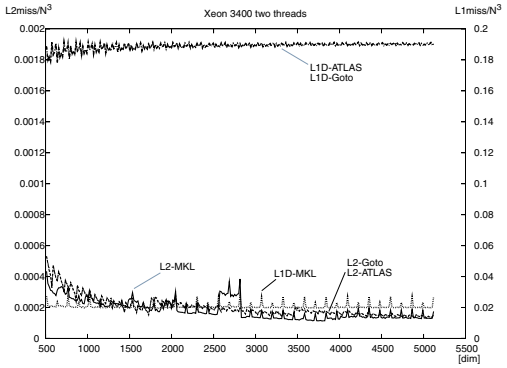

図 6 Xeon 3.4GMHz(2CPU) での dgemm のキャッシュミス率, MKL, ATLAS, Goto-BLAS の 3 種にて測定.

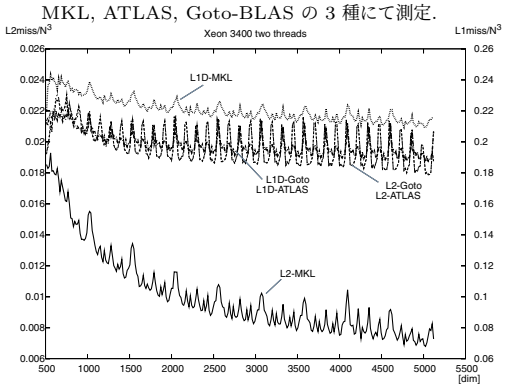

図 7 Xeon 3.4GMHz(2CPU) での dsymv のキャッシュミス率, MKL, ATLAS, Goto-BLAS の 3 種にて測定.

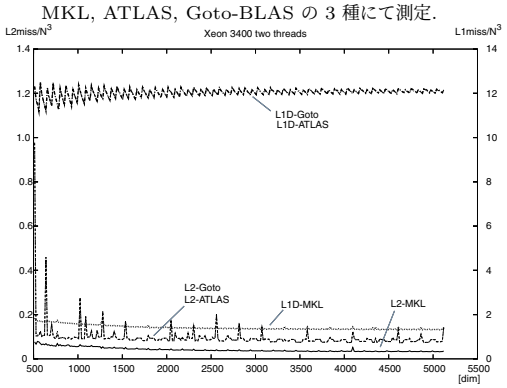

図 8 Xeon 3.4GMHz(2CPU) での dsyr2k のキャッシュミス率, MKL, ATLAS, Goto-BLAS の 3 種にて測定.

[測定項目の詳細]

- PAPI_L2_TCM (L2 キャッシュのミスヒット総数).
- PAPI_L1_DCM (L1D キャッシュのミスヒット総数).

これら項目に対して, さらに 3 関数 (dgemm, dsymv, dsyr2k) と 3BLAS 実装系 (Intel MKL 7.1, ATLAS 3.7.10, Goto-BLAS), スレッド数 (1 スレッド, 2 スレッド) の計 18 通りの組み合わせに対して各々16 次元毎に測定したものを演算回数のオーダ (dgemm の場合は N^3, dsymv の場合は N^2, dsyr2k の場合は mN^2. ただし $m = 64$ とした.) で規格化したものをプロットしたものが図 3~8 である.

データには他のプロセスの影響と考えられるシステムノイズがのっているものがあるが, 1 スレッドと 2 スレッドは同様の傾向を示していることが判る. ☆

傾向としては, ATLAS と Goto-BLAS はキャッシュ

☆ データは単一 CPU が返すカウンタ値を正規化したものであり, システム全体のキャッシュミスを測定したものとはいえない. 単純に 2 倍したものがシステム全体に相当するかは検証が必要.

ミスヒットがほぼ同じ値を示しており, 開発の母体となったコードが近いことが予想される. また, MKL は他の 2 実装比べてキャッシュミスの発生率が低く最速の実装系である理由が良く理解できる. しかしながら, 周期的なミスヒットのパルスが存在しており, それが図 1, 図 2 で観測された性能の落ち込みと見事に合致しており, キャッシュミス (スラッシング) に起因する性能劣化が発生していることがわかる.

序章にも記したが, ベンダー提供の数値計算ライブラリは「性能安定性」という観点からは十分なものとはいえないことがはっきりする. また, ベンダライブラリを組み込み関数として自動チューニングライブラリを構成する場合に, 性能劣化する次元がサンプル点として選択された際のサンプリングの信頼性に大きな疑問が生じる. サンプリングデータ空間の総数に対して性能劣化点はごく少数ではあるが, それが起こる条件や回避策などをベンダーは例示する必要があるとともに, 関数内部でそれらを積極的に回避する仕組みを導入すべきであるということが我々の主張である.

3. C-Stab アルゴリズムの評価

本章では C-Stab アルゴリズムを PAPI によって定量的な評価を行う. 大量のデータをグラフ例示するため, 結果が見にくい恐れがあるがご容赦願いたい.

3.1 C-Stab アルゴリズム

次式で定義する, キャッシュ上の位置関係から決定される相対的な距離 (位相) を用いて, まずキャッシュの不安定性が生じるかの判別を行う 12).

$$[\mathrm{I}]_L := \mathrm{mod}(\mathrm{I} + 2^{L-1}, 2^L) - 2^{L-1} \quad (1)$$

$$^*\mathrm{A} := \&\mathrm{A}/2^d \quad (2)$$

ここで, &A は A のアドレスを示し, L, s, d は個々のキャッシュの仕様によって決定される定数である.

今, n-way set associative キャッシュを搭載したプロセッサを用いたシステム上でのループ実行を仮定する. 整合寸法 (leading dimension) が N_a の 2 次元配列 a に対して, 以下の不等式が成立するとき,

$$0 < \exists i < k/n,\ |[i * N_a]_L| < \delta, \quad (3)$$

配列 a の上位インデックスを k 段展開するアンロールは, 実行時にセット競合を生じさせる. 上記判別式が成立する場合には配列のレイアウトを変更し, 決定的なキャッシュの不安定性要因を取り除き関数に渡す前処理を行う. 本アルゴリズムを C-Stab アルゴリズムと呼ぶ (詳細は文献[12]).

3.2 使用したハードウェアについて

評価に利用したハードウェア (プロセッサ) は表 1 に示すとおりである. それぞれ, ia32, em64t(AMD64 または Intel ia32 の 64bit 拡張を総称する) で代表的なプロセッサである. CPU の形態, キャッシュの構成で大きく異なることが判る. 以下, 本報告ではプロセッサを表中の第一行目に示した呼称を使って区別する.

3.3 評価プログラム

評価対象プログラムは, 我々がこれまでに開発し論文発表での計測対象であった[12] ハウスホルダー 3 重対角化ルーチンに OpneMP によるスレッド並列化を実施したルーチンと, LAPACK で同等機能を提供する `dsytrd` である. これらについて行列サイズを 16 次元毎 (但し, leading dimension と次元を一致させる) に 2 回測定し, 性能のよい方を選択した.

我々の開発ルーチンは各種最適化のためのチューニングパラメータを有するが, それらについては各プロセッサで最良なものを 1 つ選択し固定した状態で動作させた☆. また LAPACK については C-Stab での安定化を行わず, BLAS 部分に Intel MKL を利用し, ブロック幅は LAPACK ドライバが最適とした 64 を採用することとした.

使用したコンパイラは, Intel Fortran Compiler for Linux version 8.1 であり, ia32, em64t それぞれのアーキテクチャ用のものを利用した. 使用コンパイラオプションは以下のとおり.

-pc64 -O3 -tpp7 -axN -openmp (ia32 用)
-pc64 -O3 -tpp7 -axP -openmp (em64t 用).

3.4 評価結果と考察

図 9 から図 13 までに, 測定結果を示す. 測定結果から次の 3 点興味深い結果が得られた.

(1) ia32 版の MKL では 4096 次元近傍でしか性能が劣化していないが, em64t 版の MKL では X34 上で 128 次元周期, A42 上で 512 周期の性能変動が確認できる. X34 については 2 章 BLAS の評価を実施した関数 `dsymv` ならびに `dsyr2k` の結果をそのまま反映した形になる.

(2) 全てのプロセッサ, 使用スレッド数で, C-Stab アルゴリズム自体はキャッシュミスのパルスをほぼ解消している. 一部階段状の曲線が出現しているが, 発生のメカニズムについては詳細は不明である. Intel 系では L2, AMD 系では L1+L2 と異なるキャッシュ階層で起こっているため, 従来のキャッシュ安定化で考慮してきたモデルではカバーしきれていないものと考えられる. これらは今後の研究課題である.

(3) 2 スレッド使用時の振る舞いは今回使用した

☆ シングルスレッド版とマルチスレッド版では, 行列ベクトル積ルーチンの最適化で異なるパラメターが選択されている

表 1 評価に用いたシステムの詳細について

プロセッサ呼称	P26	C28	X28	X34	A42
ベンダ	Intel	Intel	Intel	Intel	AMD
プロセッサ名	Pentium 4	CerelonD	Xeon-DP	Xeon-DP	Athlon64x2
コア (所謂コードネーム)	Northwood	Prescott	Prestonia	Nocona	Toledo?
CPU 周波数 [Hz]	2.6G	2.8G	2.8G	3.4G	2.2G
CPU 形態	single	single	dual CPU	dual CPU	dual core
L1D/(CPU,core) [B]	8K	8K	8K	16K	64K
way	4	8	4	8	2
L2/(CPU,core) [B]	512K	256K	512K	1M	512K
way	8	8	4	8	16
FSB [MHz]	800	533	533	800	1000
メモリモジュール	PC3200	PC2800	PC2800+	PC2-3200+	PC3200
搭載主記憶容量 [B]	2G	1G	1G	8G	2G

なお, Xeon プロセッサの使用メモリは buffered かつ ECC 付である.

X28,X34,A42 全てことなることが判る. 特に, X34 において我々が開発したルーチンの L2 ミスが最も多いにも関わらず, MKL よりも高速であった. X28 では L1 ミスの振る舞いが X34 と同等ではあるものの, L2 では 2 スレッド版の方が 1 スレッド版のほぼ半分になっている. A42 では L1,L2 共に 2 スレッド版は 1 スレッド版の半分になっている. これはメモリバス帯域や構造の違い, またコア世代で異なるパイプライン段数の影響によって L2 ミスが増加した可能性がある. これも今後の研究課としたい.

4. ま と め

ハードウェアモニタリングシステム PAPI を用いて, 実際のキャッシュミスを測定し, C-Stab アルゴリズムがキャッシュミスを低減していることが確認された. 実際にハードカウンタによって CPU 内部の挙動を見ることによって, これまでによくわからなかった性能不安定性の原因究明の手助けになる可能性がある. 実際, PAPI 開発のテネシー大学では PAPI を用いたチューニングを開始しており, CPU 内部のより深い情報をもとに性能チューニングを行うことを今後の課題としたい. また, 今回の測定で十分な説明を与えることのできなかった現象についても, 今後究明していきたい.

参 考 文 献

1) Lawson, C., Hanson, R., Kincaid, D., and Krogh, F.: Basic Linear Algebra Subprograms for Fortran Usages, *ACM Transacton on Mathematical Software*, Vol. 5, pp. 308–323 (1979).
2) MKL : Intel Kernel Library, `www.intel.com/software/products/mkl/`.
3) ACML : AMD Core Math Library, `http://developer.amd.com/acml.aspx` .
4) Whaley, R.C., Petitet, A., and Dongarra, J.J.: Automated Empirical Optimization of Software and the ATLAS Project, LAPACK Working Note 147 (2000).
5) Goto, K. and Van de Geijn, R.: On Reducing TLB Misses in Matrix Multiplication, FLAME Working Note #9, *Technical Report of The University of Texas at Austin, Dept. of Computer Science*, TR-2002-55 (2002).
6) Bilmes, J., Ananoic, K., Chin, C.-W., and Demmel, J.: Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology, *Proceedings of International Conference on Supercomputing 97*, pp. 340–347 (1997).
7) 片桐孝洋, 黒田久泰, 大澤清, 工藤誠, 金田康正: 自動チューニング機構が並列数値計算ライブラリに及ぼす効果, 情報処理学会誌：ハイパフォーマンスコンピューティングシステム, Vol.42, No.SIG12(HPS 4), pp. 60–76 (2001).
8) Katagiri, T., Kise, K., Honda, H., and Yuba, T.: FIBER: A Generalized Framework for Auto-tuning Software, *Proceedings of 5th International Symposium, ISHPC2003*, LNCS 2858, pp. 146–159 (2003).
9) Pettersson, M.: Perfctr, the Linux performance monitoring counters kernel extension `http://user.it.uu.se/mikpe/linux/perfctr/`.
10) London, K., Moore, S., Mucci, P., Seymour, K., and Luczak, R.: The PAPI Cross-Platform Interface to Hardware Performance Counters, Department of Defense Users' Group Conference Proceedings (2001).
11) 今村俊幸, 直野健: 性能安定化を目指した自動チューニング型固有値ソルバーについて, 先進的計算基盤システムシンポジウム SACSIS2003, pp. 145–152 (2003).
12) 今村俊幸, 直野健: キャッシュ競合を制御する性能安定化機構内蔵型数値計算ライブラリについて, 情報処理学会論文誌, コンピューティングシステム, Vol. 45, No.SIG 6(ACS 6), pp. 113–121 (2004).

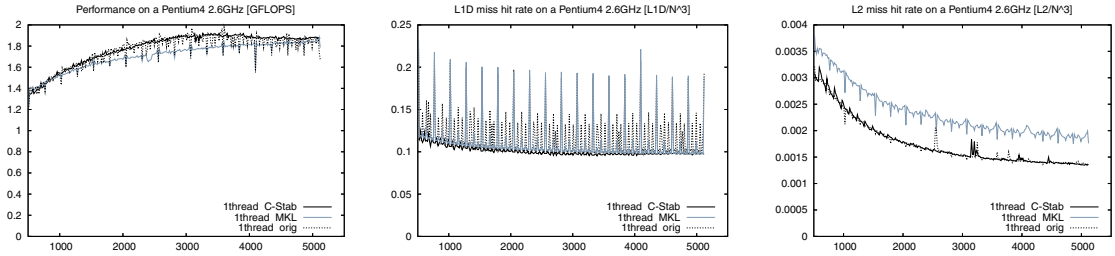

図 9 P26 の性能評価 (左:FLOPS, 中:L1D miss, 右:L2 miss)

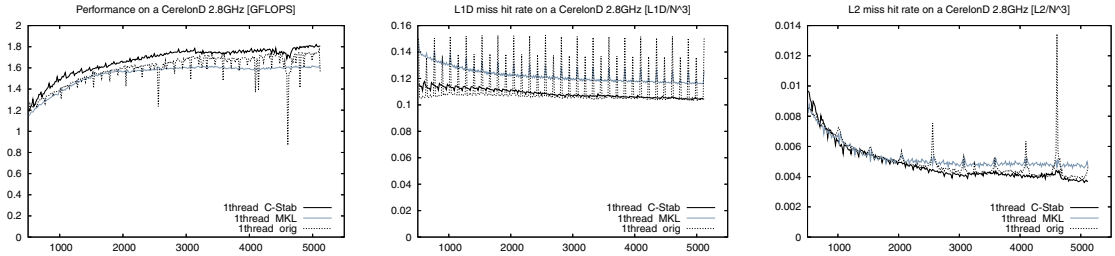

図 10 C28 の性能評価 (左:FLOPS, 中:L1D miss, 右:L2 miss)

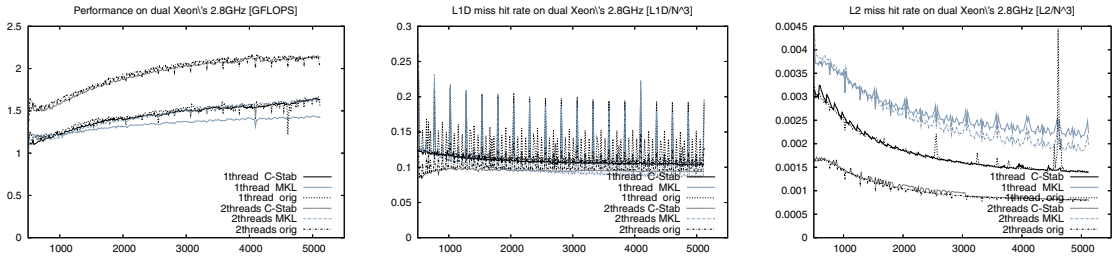

図 11 X28 の性能評価 (左:FLOPS, 中:L1D miss, 右:L2 miss)

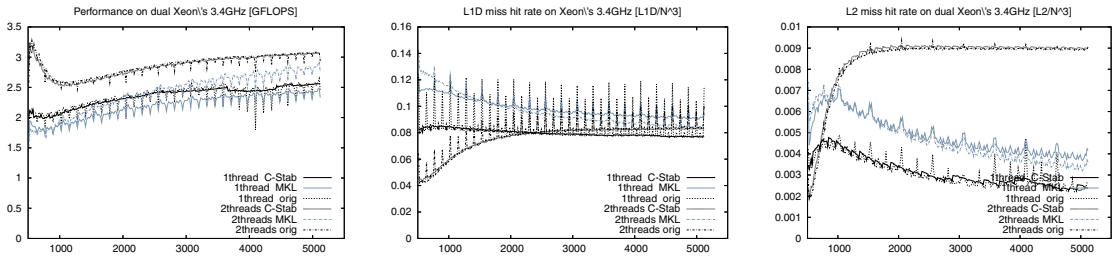

図 12 X34 の性能評価 (左:FLOPS, 中:L1D miss, 右:L2 miss)

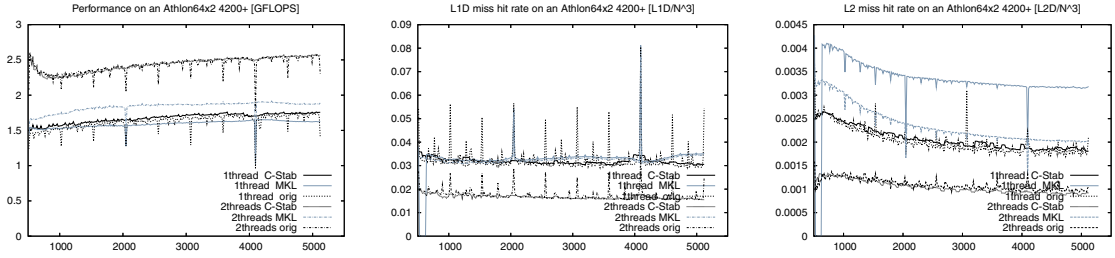

図 13 A42 の性能評価 (左:FLOPS, 中:L1D miss, 右:L2 miss)