GotoBLAS チュートリアル

後藤　和茂 (テキサス州立大学)

お題目

- 自己紹介
- 数値計算と最適化の基本事項の確認
- BLAS とは？
- GotoBLAS の特徴
- Level 1 ～Level 3 ルーチンの構造と特徴
- BLASによる最適化の限界

自己紹介

- 早稲田大学電気工学修士課程卒
- 特許庁で審査官（10年ほど）
- 数値計算は実は趣味だった
- 留学の許可→受け入れ先を探した
- 多数のメールを出すも返事は1通のみ
- テキサスでちょっと本気を出しました
- 現在はテキサス州立大学で研究職

テキサスって？

- オースチンには意外と緑がある
- 物価が安い・治安が良い
- 郊外は砂漠というより、岩だらけの不毛の地 (Cars という映画のとおり)
- 毎年異常気象（地震はないが、雹あり）
- 夏は毎日40度以上、室内は25度以下
- 冬は2週間くらい（「氷」が降ることも）
- なまりがひどい
- 何でも大きい（テキサスサイズ）

BLAS ってなに？

- Basic Linear Algebra Subprograms の略
- ベクトル及び行列に対する基本的な浮動小数点演算を行うサブルーチン群
- 標準仕様なので、各 CPU 向けに最適化された BLAS を使用することができる
- 複数の最適化 BLAS があっても、切替簡単
  - ユーザはドライなので、性能が悪いと簡単に切り捨てられる

BLAS に対する誤解

- 全てのパラメータに対して速い？
  - BLAS を呼びさえすればプログラムが速くなると思っている人が多すぎる
  - 実際にはかなり厳しい条件を満たす必要あり
  - 計算量が極端に小さい場合、かなり遅くなる
- BLAS はブラックボックスとみなしてよい？
  - どのような原理で動作しているのかを理解する必要あり

どの最適化 BLAS を使えばいいか？

- 適当に速ければいい（普通のユーザ）
  - MKL を買えば安心？（商品）
  - ATLAS (フリー)
  - GotoBLAS (学術研究向け)
- とにかく速いのが欲しい（ヘビーユーザ）
  - MKL か GotoBLAS
- BLAS を利用して新規のアルゴリズムを開発したい（学術向け）
  - GotoBLAS

GotoBLAS の特徴 (1)

- 複数の最新～最古のアーキテクチャに対応
  - Intel Core 2
  - Intel Pentium4, Pentium3
  - Intel Itanium2
  - AMD Opteron, Athlon
  - IBM POWER5, BG/L, QCDOC
  - SUN SPARC IV
  - Alpha EV4,EV5,EV6

GotoBLAS の特徴 (2)

- 共通のインタフェース及びアルゴリズムを使用(重要！)
  - あるアーキテクチャで良好な性能が出るならば、他のアーキテクチャ上でも性能は良い
- 異なるのは
  - m, n に関するアンローリング
  - ブロッキングサイズ
    のみ

GotoBLAS の特徴 (3)

- 開発期間が短い
  - 倍精度行列積のみだと3～7日前後
  - BLAS全体で1ヶ月程度
- 性能がそれなりに良い
  - できるだけ汎用性を持たせて作ってある
  - 細かい部分ではベンダ製ライブラリより劣る
- ソースコードが参照できる
  - 肝心な部分はアセンブラなので意味無いかも？

GotoBLAS の特徴 (4)

- データの配置・移動が立体的である
  - A のデータは L2 上、B のデータの一部は L1 上等
- Bandwidth aware なライブラリ
  - バンド幅の要求量をコントロール
  - バンド幅が狭くてもそれなりの性能が出せる
    - POWER5 上で性能が良いのは当たり前
    - PPC970 上でも性能が良いならば本物

GotoBLAS の設計目標

- 速やかに最適化 BLAS を提供すること
  - ベンダのライブラリは十分な最適化が行われていなくて、見切り発車的なケースも多い
- 究極的な性能は追求しない
  - 性能の上限を正確に予測することにより見切りをさっさとつける
  - 性能には上限があるので、数ヵ月後にベンダに追いつかれても気にしない
  - よほどのことがない限り、この上限以上に性能が良くなることはない

BLAS の最適化の基本方針

1. CPU の浮動小数点演算処理能力（これもバンド幅の一種）が律速となるようにする
2. キャッシュ、メモリのバンド幅が律速となるようにする
   - SMP でのスケーラビリティは悪くなる
3. 最適化の要はバンド幅の制御にあり
4. ちなみに、レイテンシは隠すものであって、見せるものではない

メモリに関する用語集

- 大きさ(大きい、小さい)
- バンド幅(広い、狭い)
  - 単位時間で転送できるデータの量
  - 大きいほうが高性能
- レイテンシ(大きい、小さい)
  - データの要求を行ってから実際にデータが転送されるまでの待ち時間
  - 小さいほうが高性能

バンド幅が広くレイテンシが小さいメモリが良い

キャッシュとは？

- メインメモリは容量は大きいが、バンド幅及びレイテンシの特性は悪い
- 一度使用されたデータ、隣り合うデータは再利用される可能性が高い
- 高速なメモリを使って使用されたデータを保持しておく
- 特性の異なったキャッシュを組み合わせて階層構造にする

キャッシュの特性で一番重要なのは？

1. 大きさ（Size）
2. バンド幅 (Bandwidth)
3. レイテンシ (Latency)

１．バンド幅、２．レイテンシ、３．大きさ

実は、大きさというのはあまり重要ではない
（メーカーの宣伝に騙されてはいけない）

キャッシュの本性

- 理論通りにキャッシュ上にデータは存在しないことが多い
  - 物理アドレスの管理が重要
    - OS によって特性が異なる。Tru64 最良, OSX 最悪
  - 一般に有効なキャッシュサイズは実際の半分
- 同時アクセスの制限
  - バンクコンフリクトの回避（最高難度）
  - キャッシュミスが発生した際の優先度の違い

FORTAN (Column major) での配列

- 列方向へのアクセスは、メモリ上でも連続したアクセスになる
- 行方向へのアクセスは致命的
  - メモリ上を飛び飛びにアクセスする
  - TLBミスが頻繁に発生する
  - キャッシュミスも発生する

incx, incy というパラメータでは
出来る限り1 を指定する

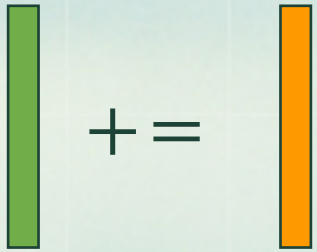
BLAS の精度

	単精度 (32bit)	倍精度 (64bit)	拡張倍精度 (128bit)
実数	S	D	Q
複素数	C	Z	X

例： SGEMM, DTRSM, CSYMM, ZSYRK

BLAS Level 1

- ベクトル単体、もしくは同士の演算



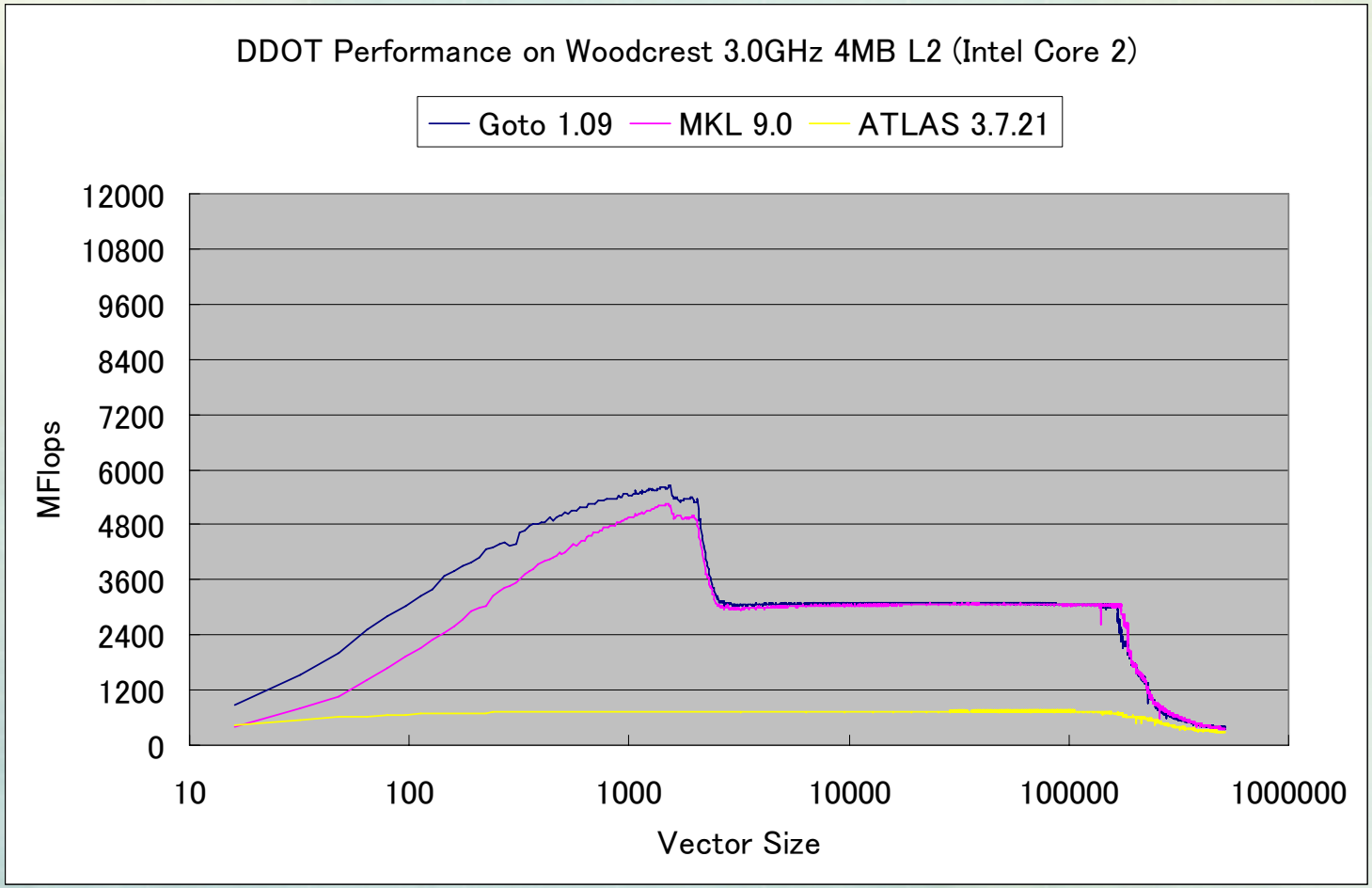
- データの再利用性はない
  - データがキャッシュ上にある場合には高速
- 演算量が少ない (m)
  - 命令サイズはできるだけ小さく
- 性能はCPUの理論性能値、またはメモリバンド幅に依存

BLAS Level 1 主な関数

- DOT　　：内積演算
- AXPY　　：乗算＋加算
- NRM2　　：ノルム
- COPY　　：コピー
- SWAP　　：交換
- I?AMAX　：絶対値が最大の要素の検出

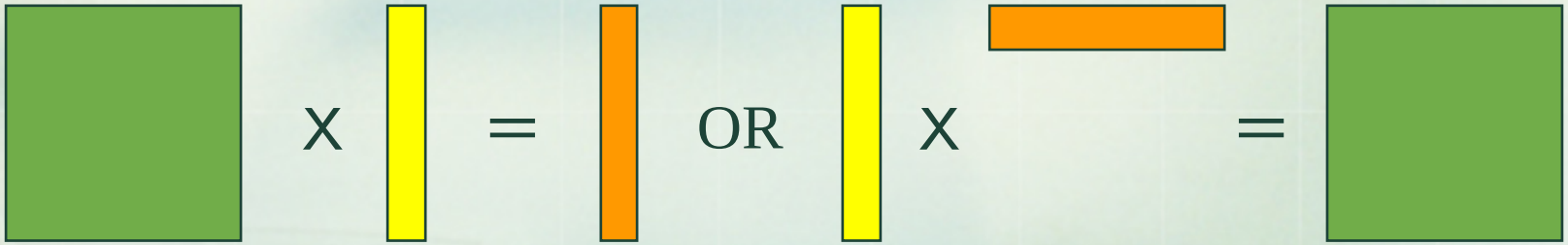
Incx, Incy というパラメータがあるが、1 以外は最適化が行われていない

BLAS Level 1 の典型的な特性



BLAS Level 2

- 行列とベクトルの演算

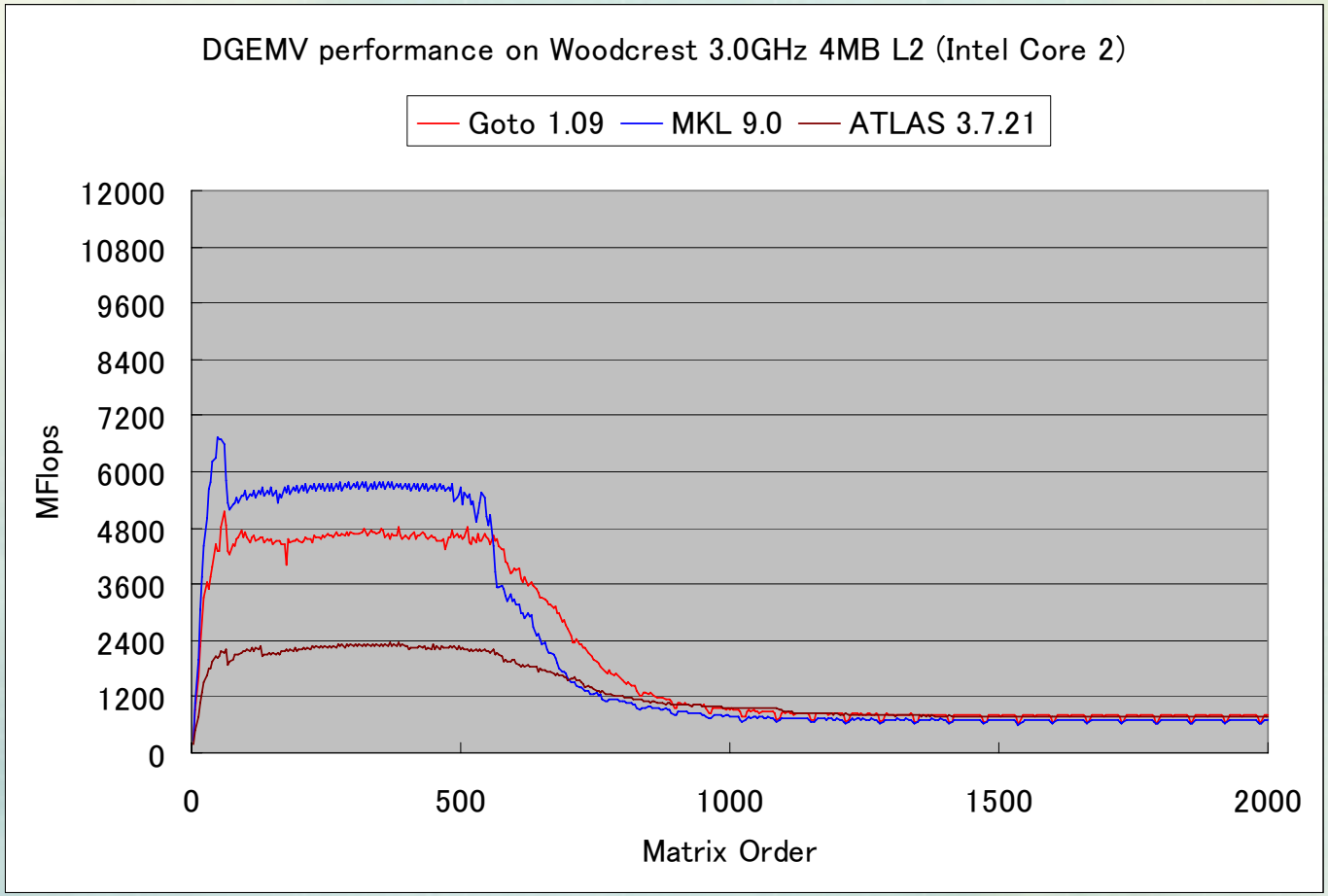


- ベクトル部のみデータの再利用性あり
  - 行列部のアクセスで大量のキャッシュミス
- 演算量は程々に多い (m x n)
  - 命令サイズは大きくても可
- 性能はメモリバンド幅に依存

BLAS Level 2 主な関数

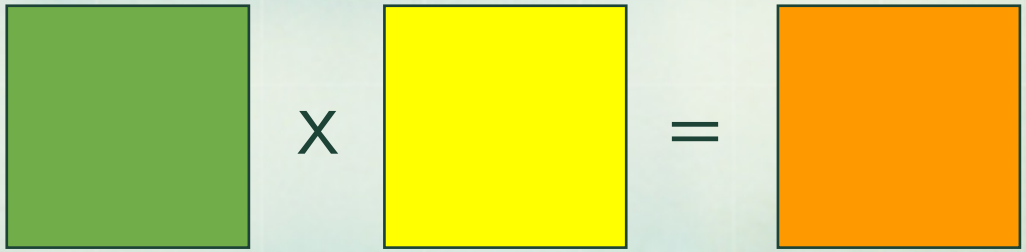
- GEMV　：行列ーベクトル積
- GER　：Rank-1 更新
- TRMV　：三角行列ーベクトル積
- TRSV　：三角行列ーベクトル求解
- SYR　：対称行列の Rank-1 更新
- SYR2　:対称行列の Rank-2 更新
- GBMV　：バンド行列ーベクトル積

BLAS Level 2 の典型的な特性



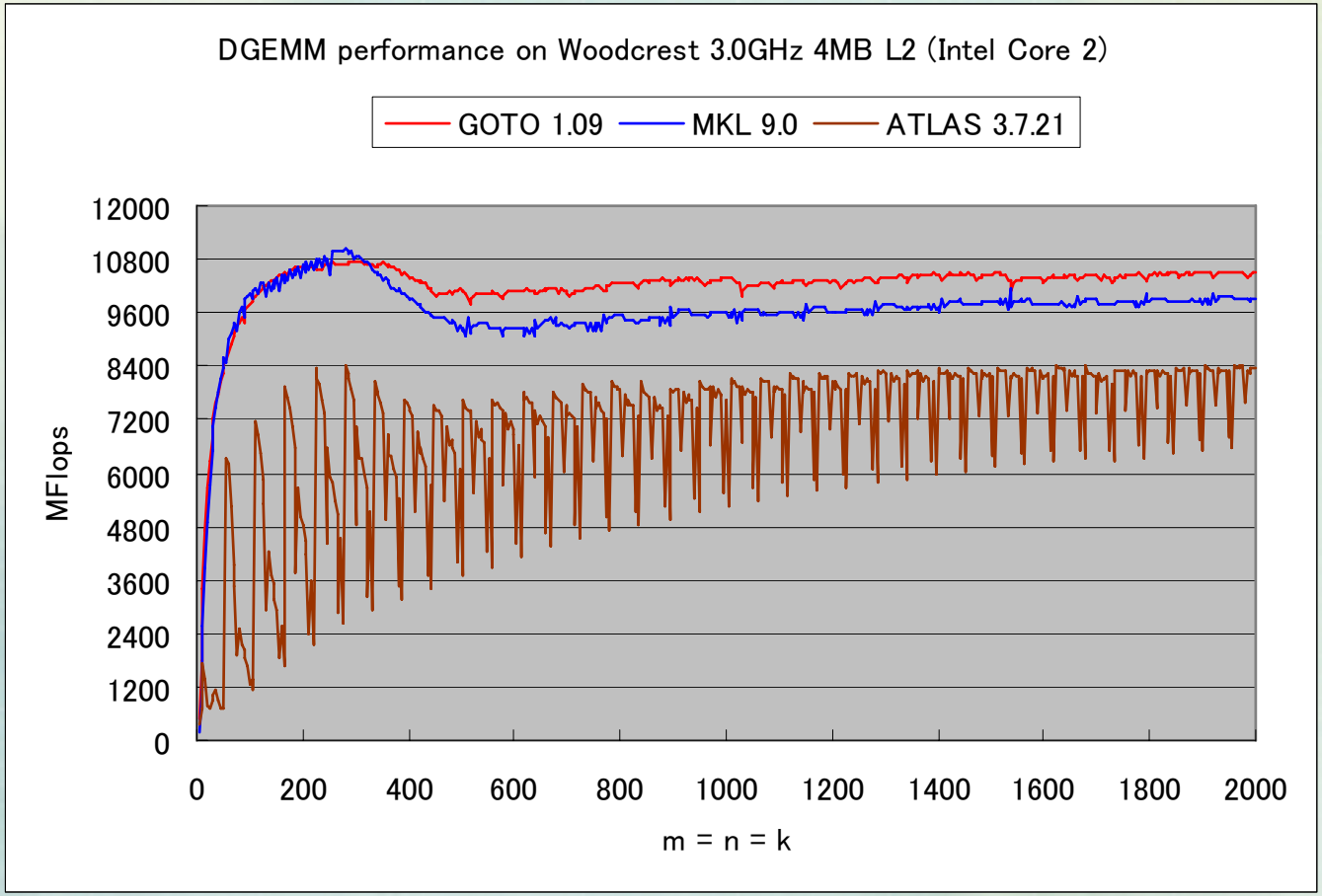
BLAS Level 3

- 行列同士の演算



- データの再利用性大
- 演算量は極めて多い (m x n x k)
  - 命令サイズの制限はかなり緩い
- 性能は CPU の理論性能値に依存

BLAS Level 3 の典型的な特性



行列積での最適化手順

1. 基本アルゴリズムの変更はない
   - 演算量は同一
2. アクセスパターンを利用したブロッキング
   - バンド幅コントロール
   - コピーによるパッキング処理
3. カーネルチューニング
4. 性能予測値との比較（必要なら3へ戻る）

従来と異なる点

- 最適化を行う前に既に決まっている事項
  - m 及び n に関するアンローリング
  - ブロッキングサイズ
  - 予測性能値
- 最適化作業中に判明する事項
  - CPUの特性から由来する性能劣化
- 最適化作業は全部で3～4日程度

コピーによるパッキング

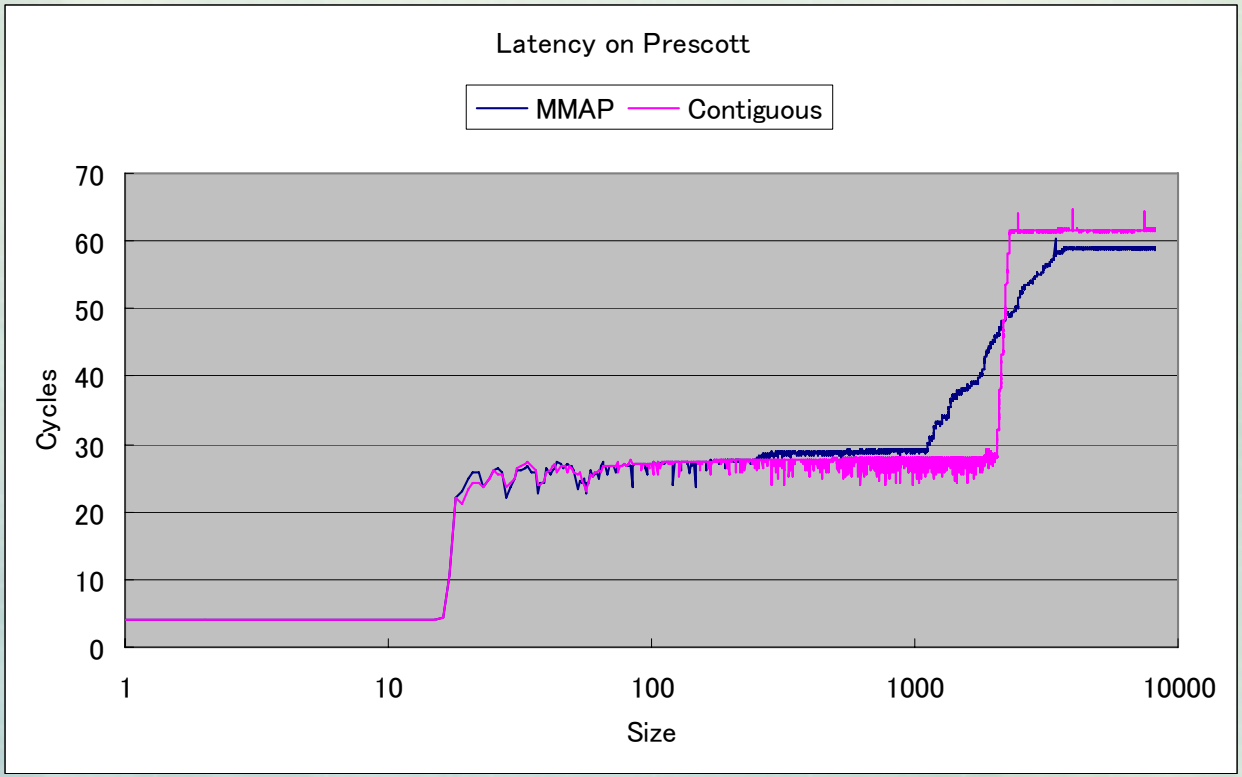
- 単純なブロッキングは問題あり
  - 先導行列(Leading Dimension) の影響大
  - SSE2 等で要求されるアライメントに対応できない
  - キャッシュラインを利用したアクセスができない
- コピーを用いたパッキング
  - コピーのオーバーヘッドの考慮
  - アクセス順序に従って1本のストリームに変換する
  - HugeTLB fs 等の特殊なメモリを使用できる

二次キャッシュの悪いクセ

- ベンチマークを実行すると
  - 速→遅→速→遅 … という現象が発生する
- 物理アドレスを連続させる必要あり
- BSD 最悪、Linux 普通、Tru64 良好
- 全部使い切るのは通常のやり方では無理
- 半分よりちょっと少なめがベスト
- どうしても全部使いたい場合:HugeTLB fs

二次キャッシュの問題点

- 二次キャッシュからのストリーミングは、カーネルのメモリ管理の影響を受けやすい



何でも速いか？

- 大きな弱点あり
- 考慮するべき点は
  - コピーのオーバーヘッド
  - C のアクセスコスト

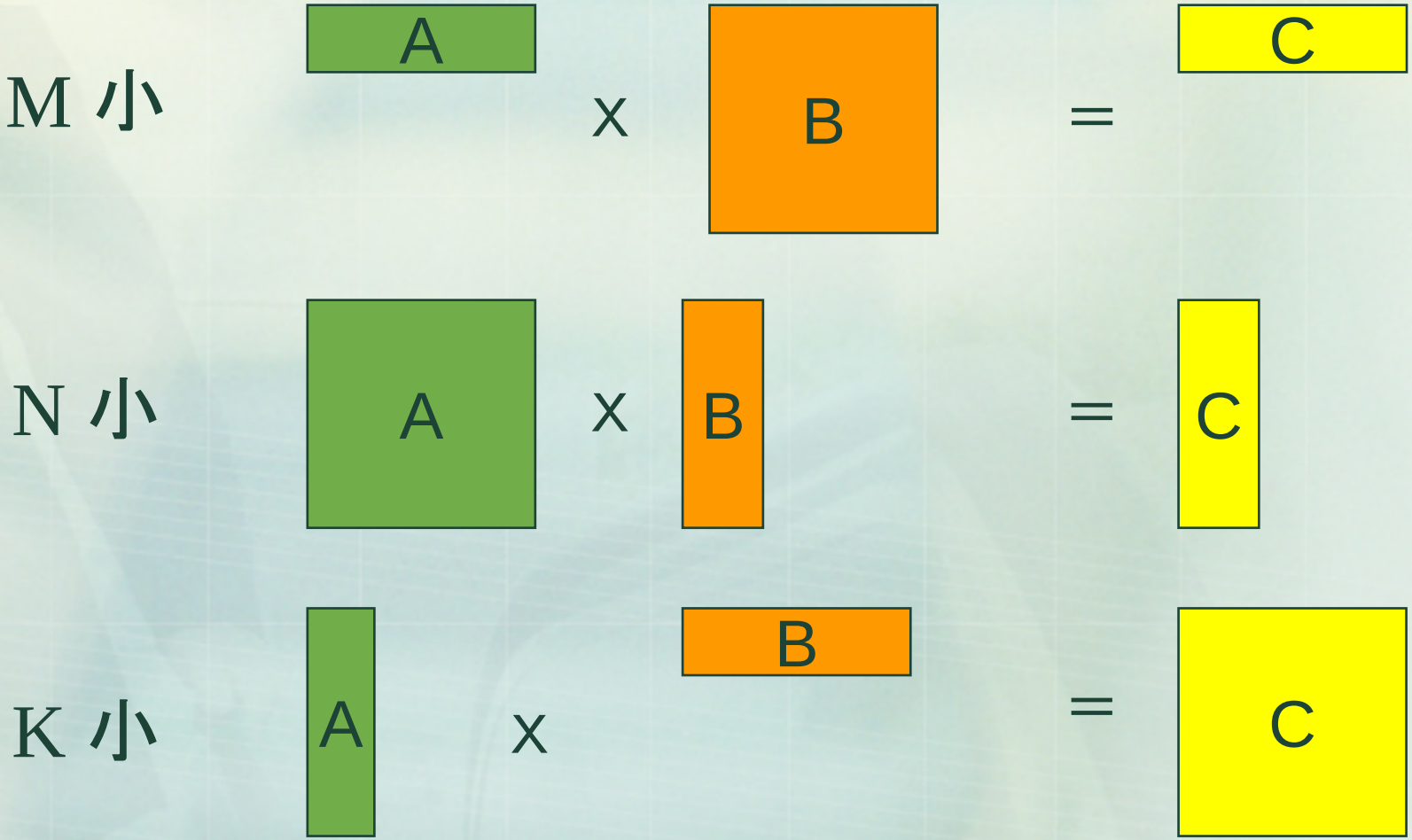
この関係を理解できると、プログラムの速度がとても速くなる

行列積ルーチンのアルゴリズム

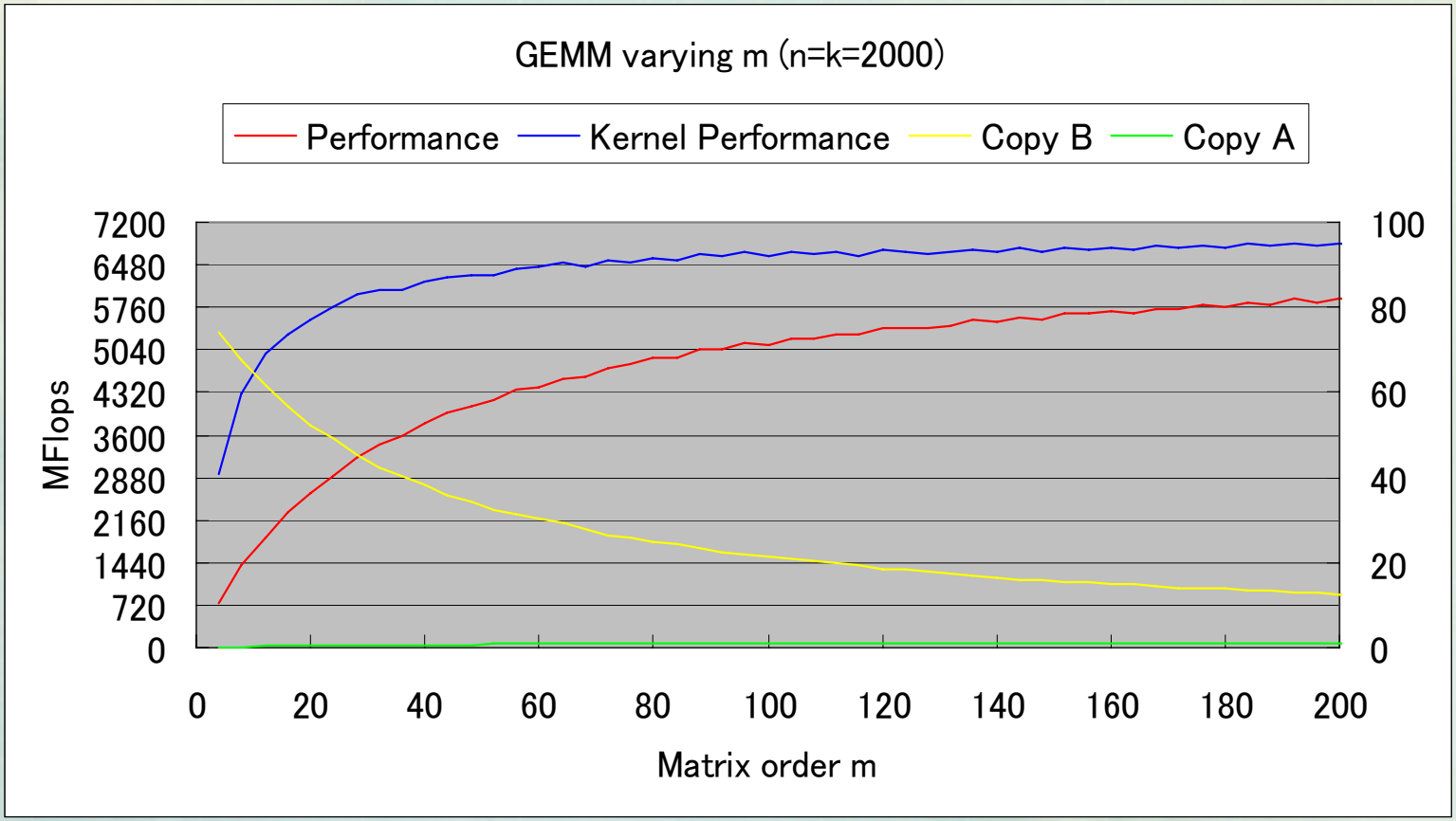
1. B の一部をコピー（B’）
2. A の一部をコピー（A’）
3. カーネルルーチンでの計算
   - A’ ＊ B’ の計算をする
   - C には直接アクセス＆更新する

コピー操作が全体の性能にどのように影響を及ぼすのか？

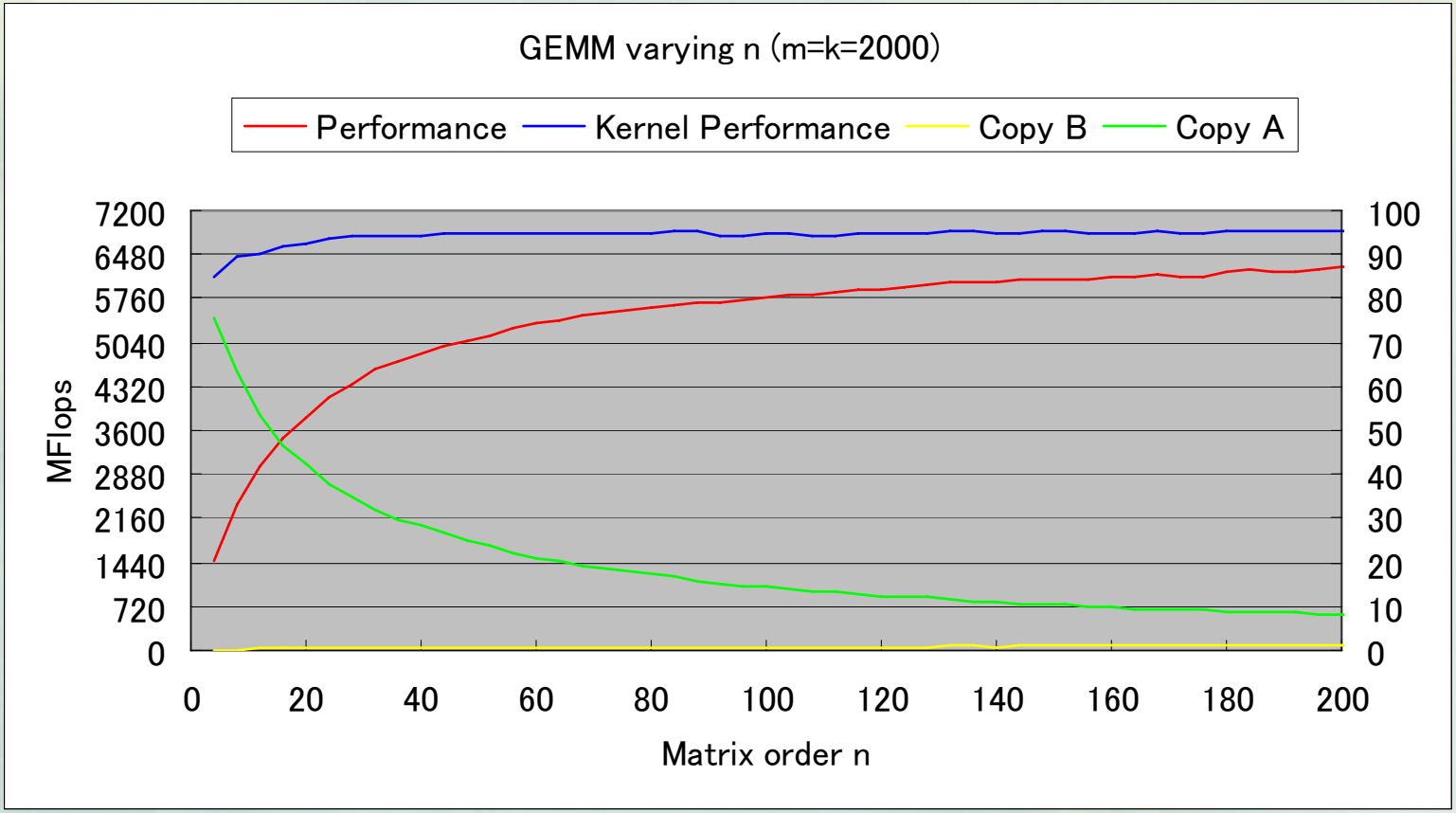
コピーのオーバヘッドの影響



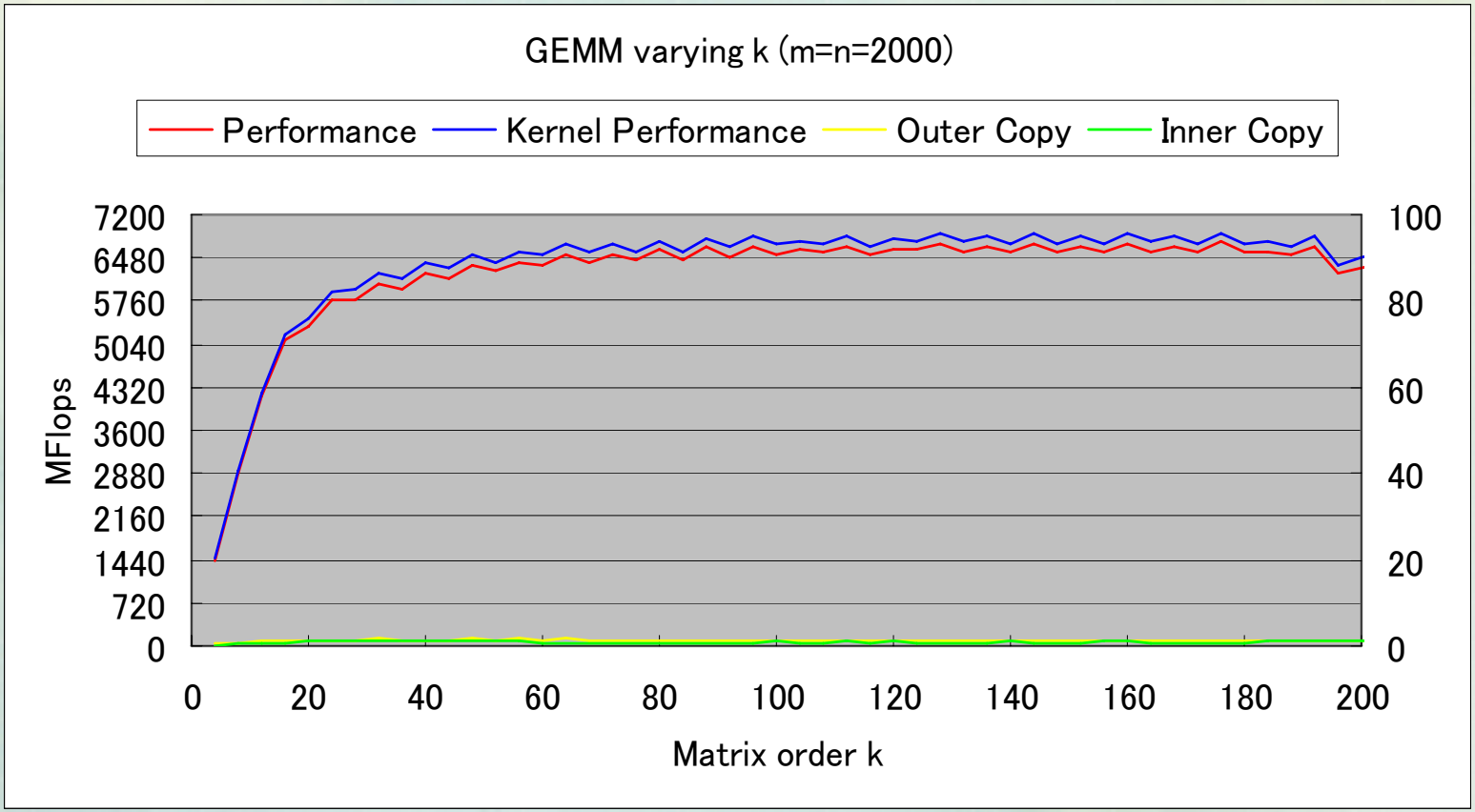
コピーオーバーヘッド m (Pentium4)



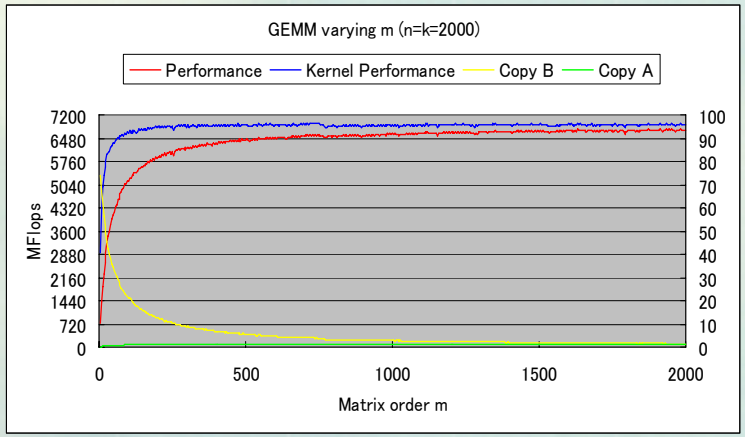
コピーオーバーヘッド n (Pentium4)

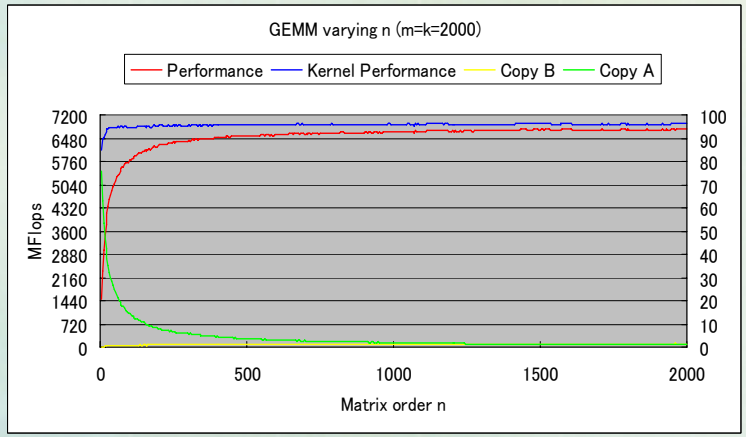


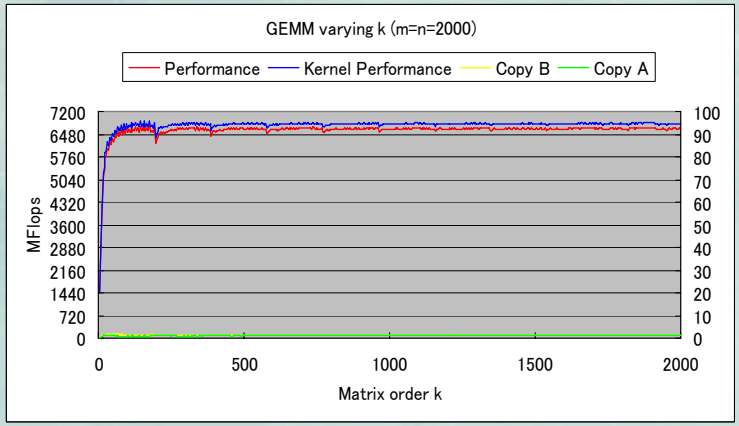
コピーオーバーヘッド k (Pentium4)

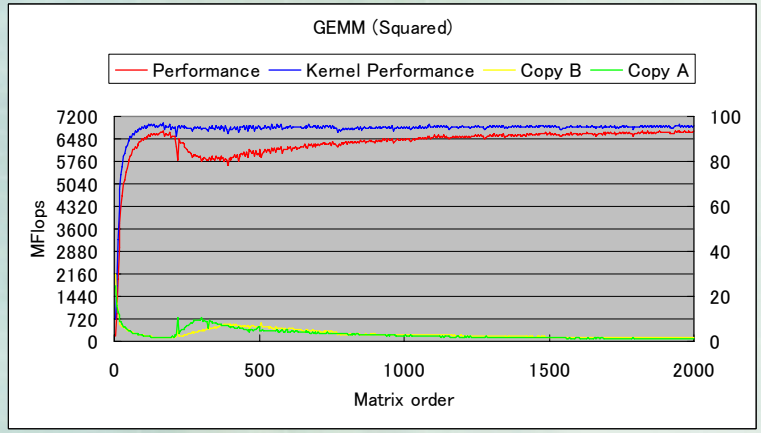


コピーオーバーヘッド (Pentium4)

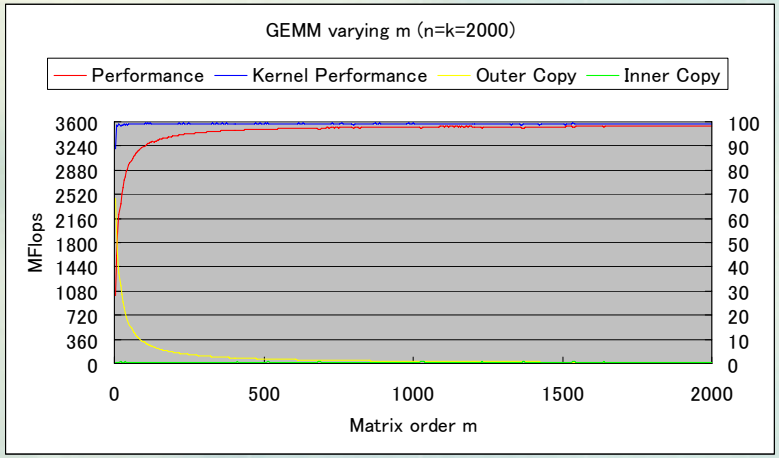


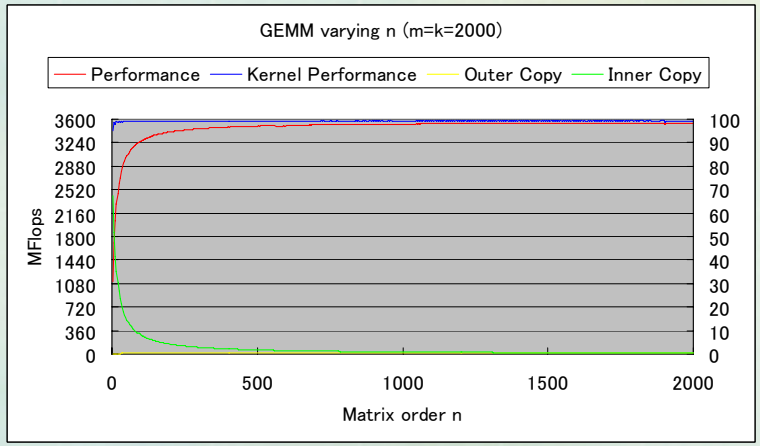


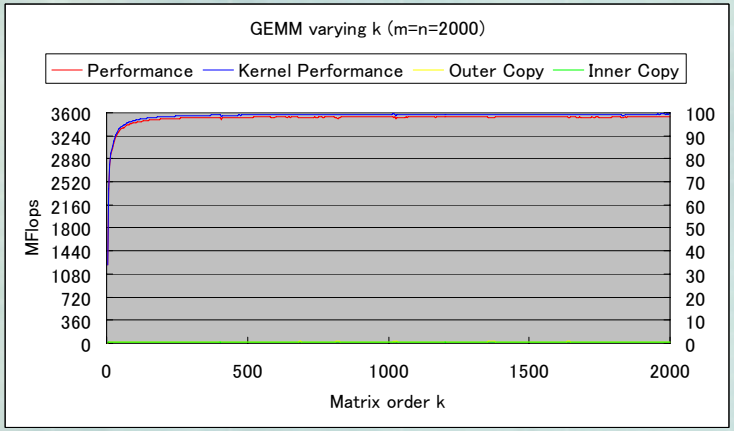


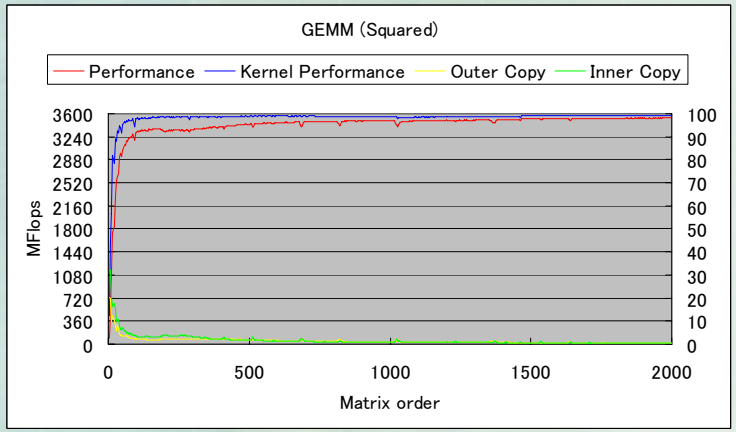


コピーオーバーヘッド (Itanium2)



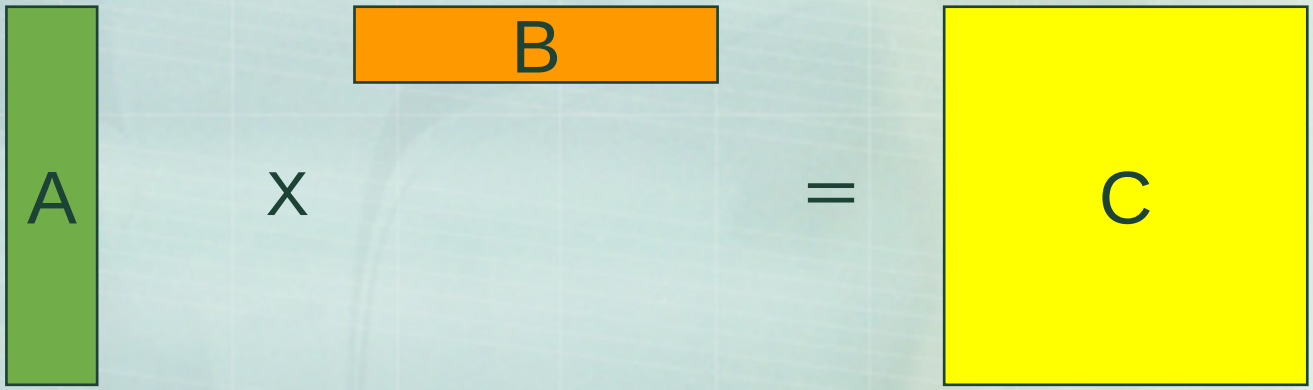






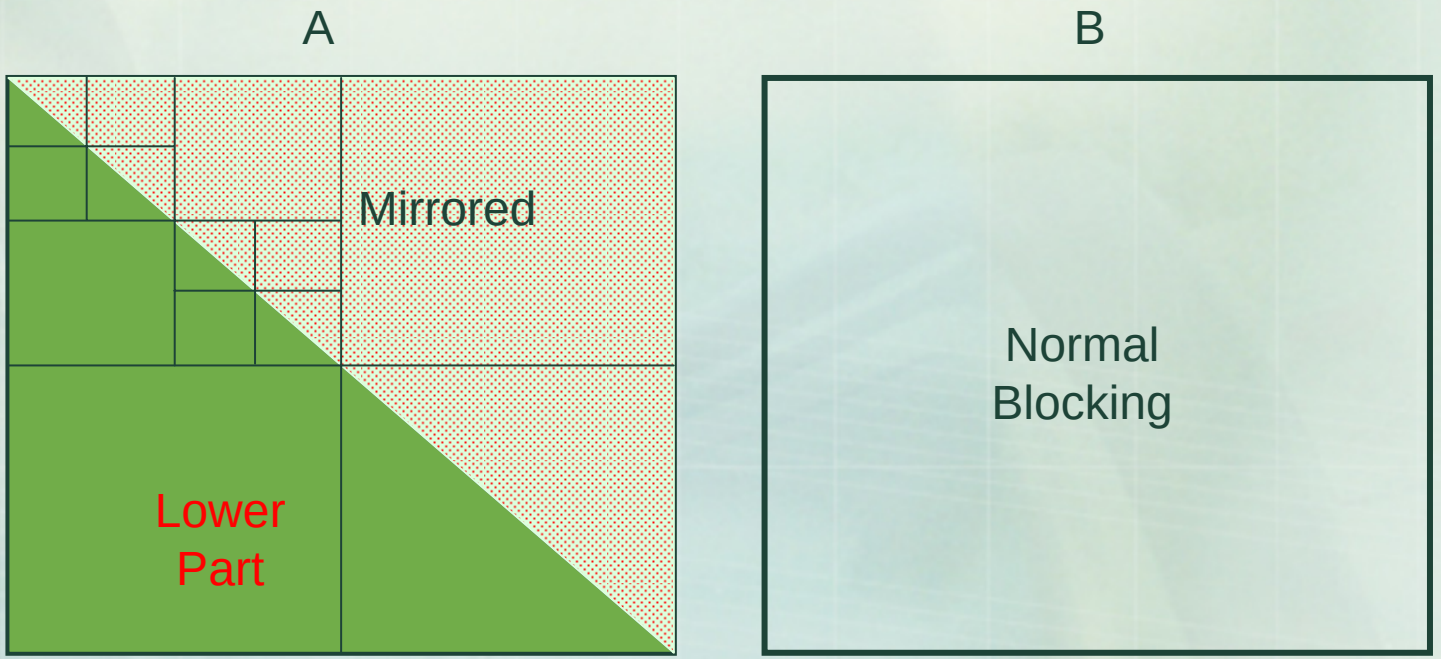
以上からわかること

- m と n はできるだけ大きい方が良い
- k はある程度大きければよい
- カーネル自体は性能は一定
- 以下の形がベスト→ Level 3 に応用



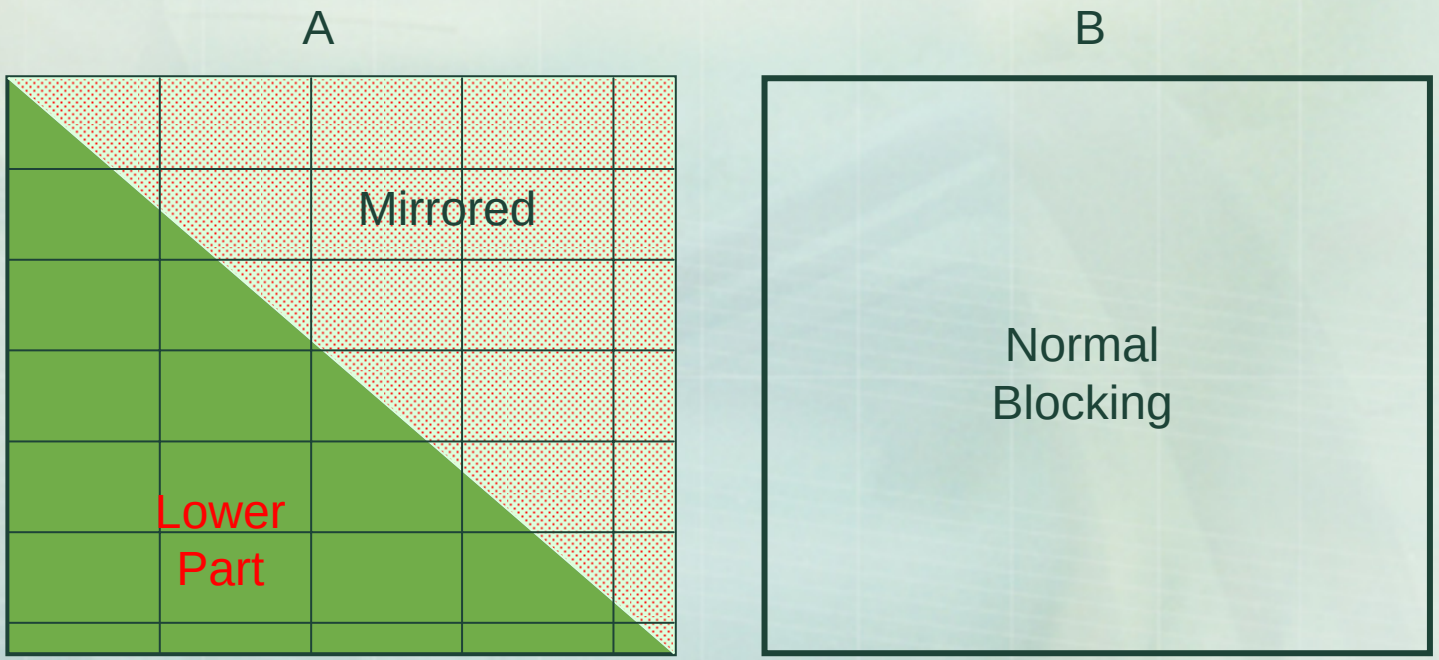
再帰 BLAS（格好いいかも）

小ブロック時のコピーオーバーヘッドが大きい

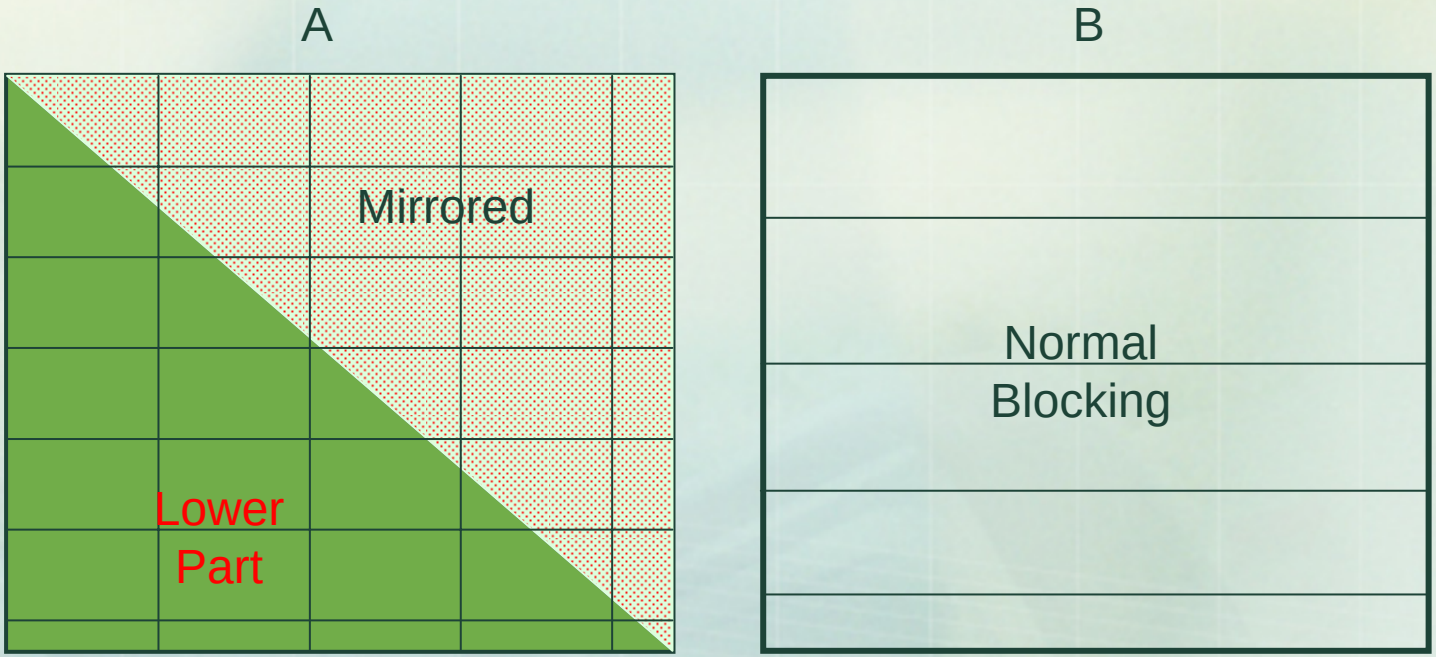


美しい再帰→地味な処理へ変更

- 行列積と同様のブロッキングを行う
- 専用のカーネルが必要

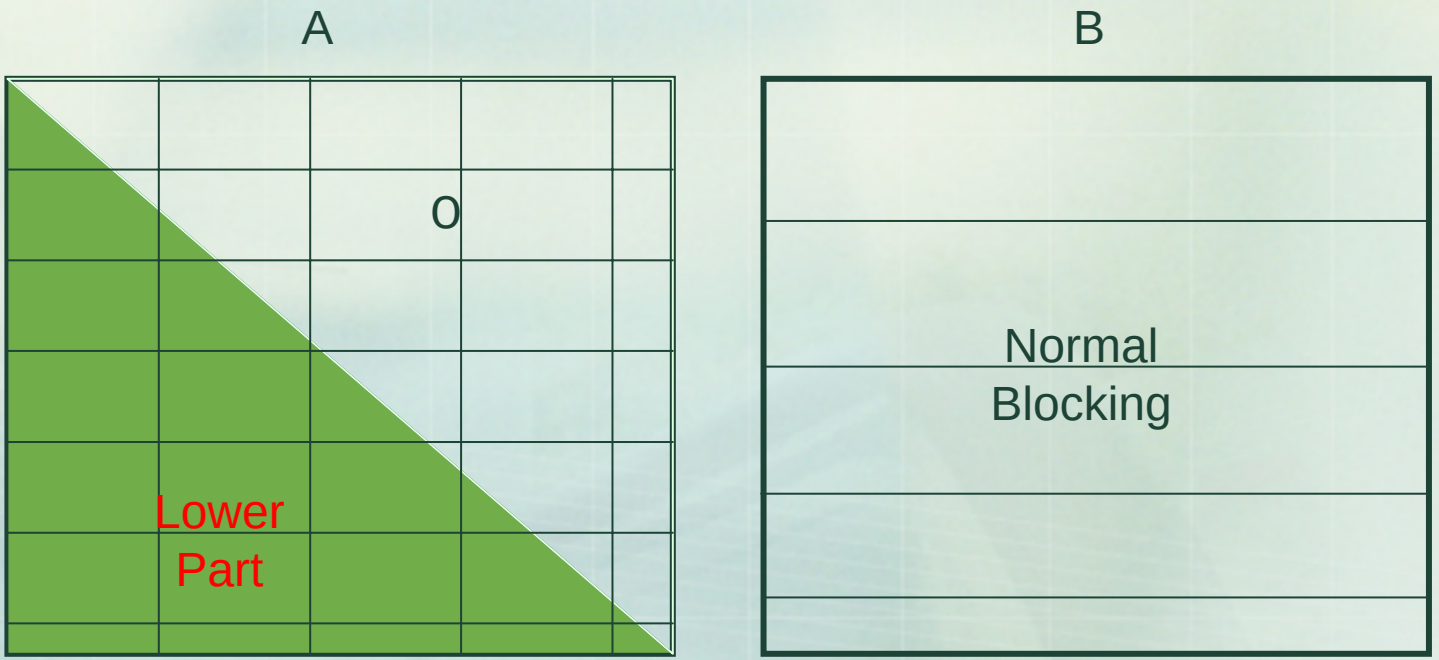


SYMM



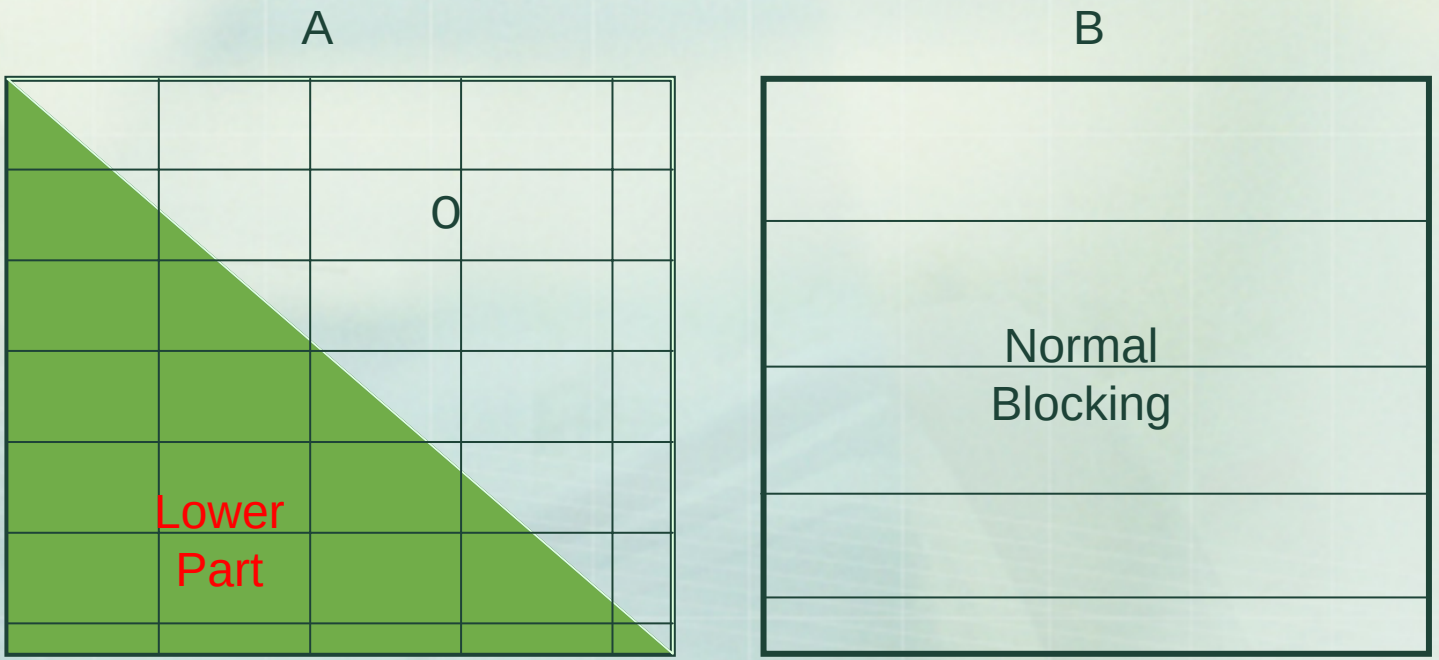
- A のコピールーチンのみ作成が必要
- SYMM カーネルは GEMM のカーネルと同一

TRMM



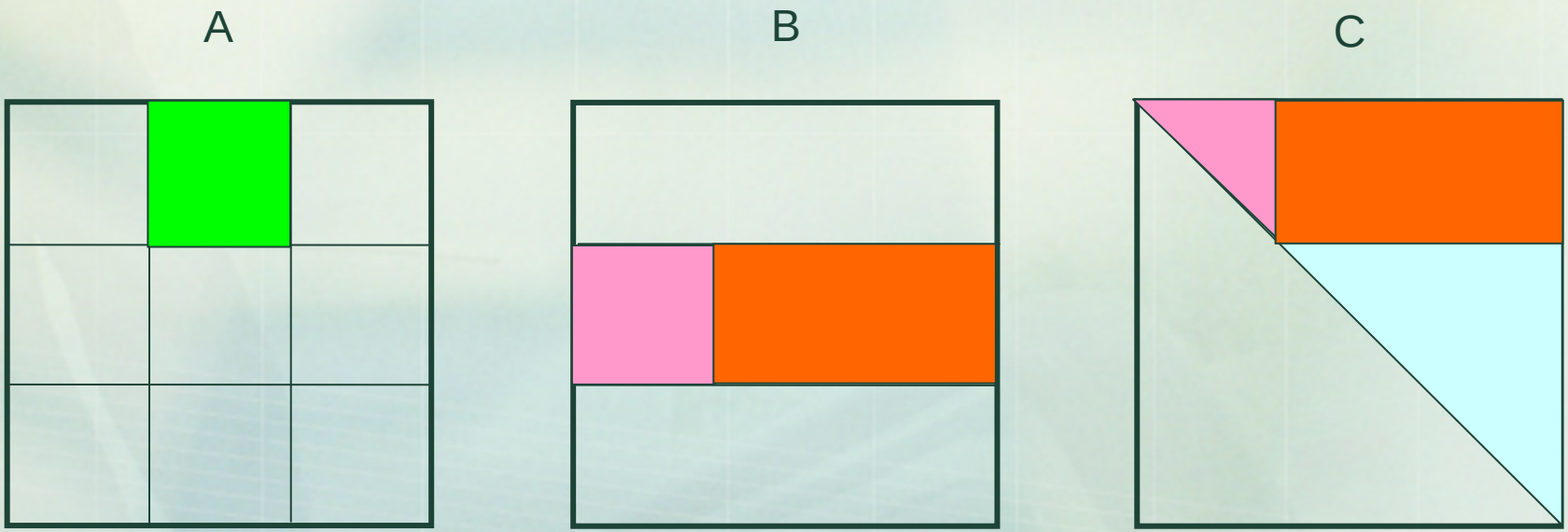
- 対角部分だけ0を挿入して TRMM カーネルで計算
- 0の部分は当然計算はスキップ

TRSM



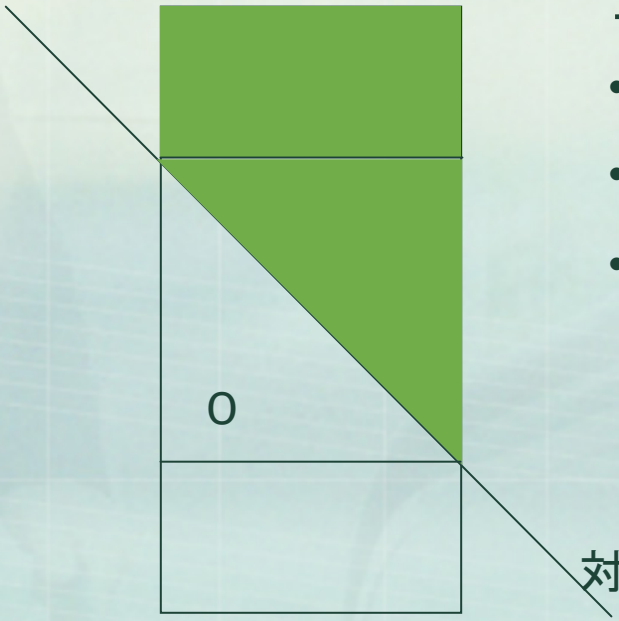
- データ依存性の関係を除いてほぼ TRMM と同等
- 依存関係のため、結果をバッファに書き込む必要あり

SYRK/SYR2K



- GEMM カーネルを少し変更するだけでよい
- 専用のカーネルは作っても構わないが、性能はそれほど向上しない

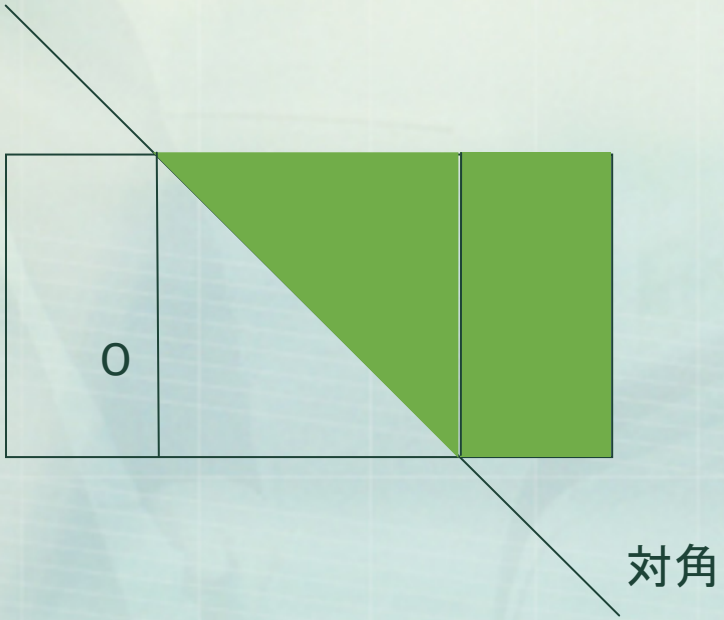
TRMM カーネルその1



全部で3つ

- GEMM カーネル
- TRMM カーネル
- なにもしない

TRMM カーネルその2



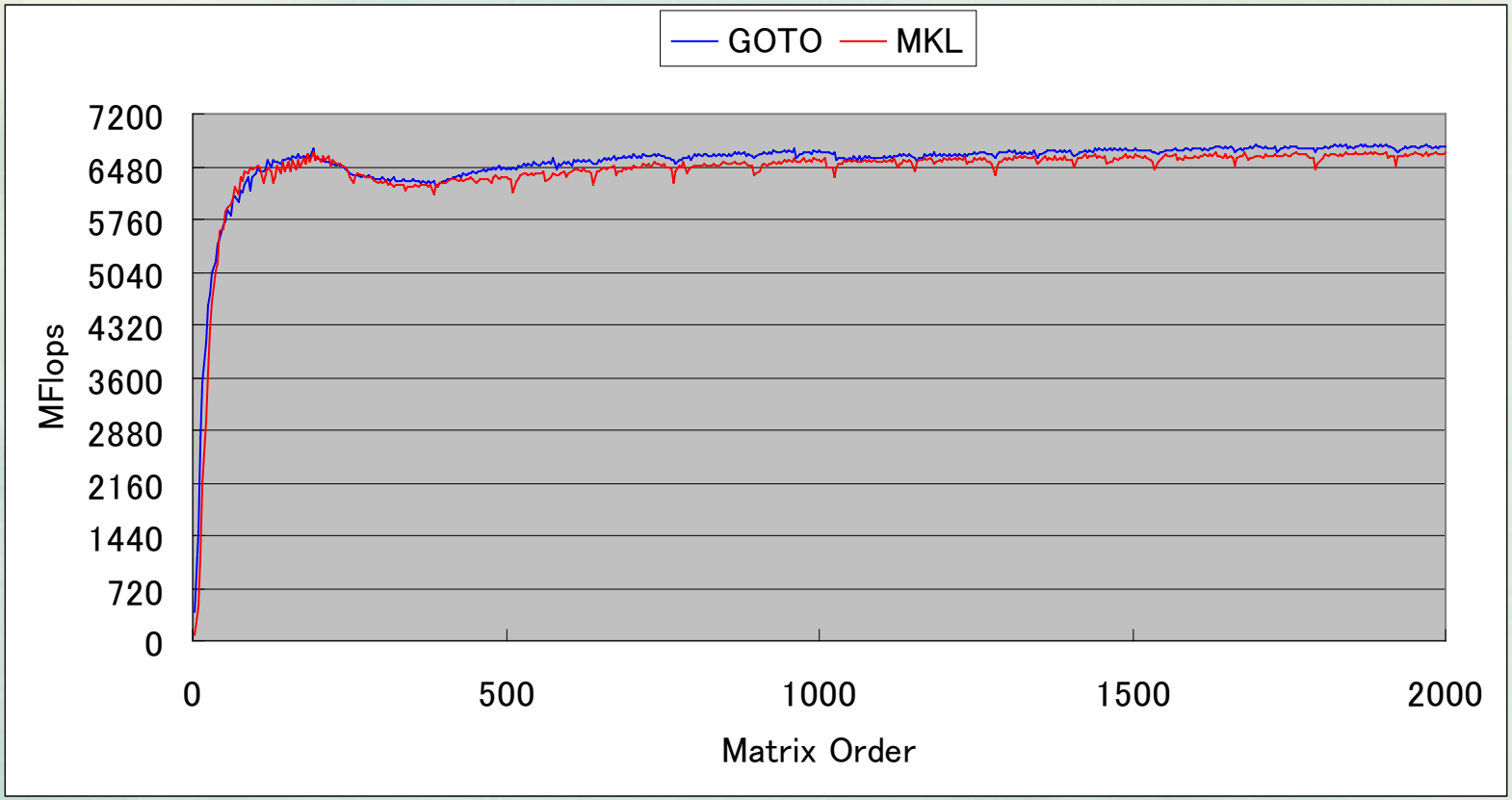
全部で3つ

- GEMM カーネル
- TRMM カーネル
- 何もしない

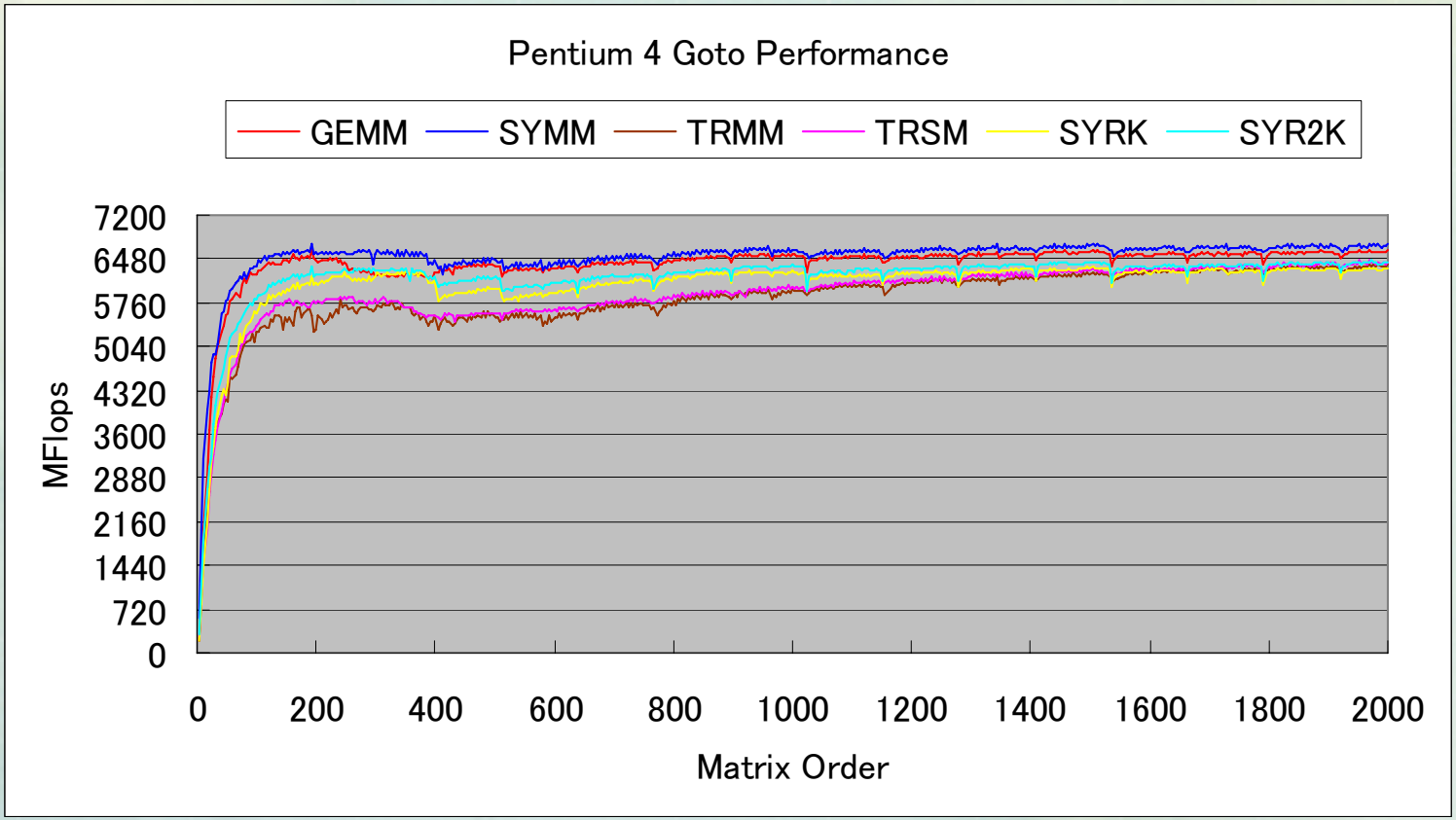
ベンチマーク

- Pentium4 Prescott 3.6GHz 2MB L2
- ピーク性能: 7.2GFlops
- コンパイラの最適化フラグ: –O2
- MKL のバージョン: 8.0.1
- ATLAS のバージョン: 3.7.11

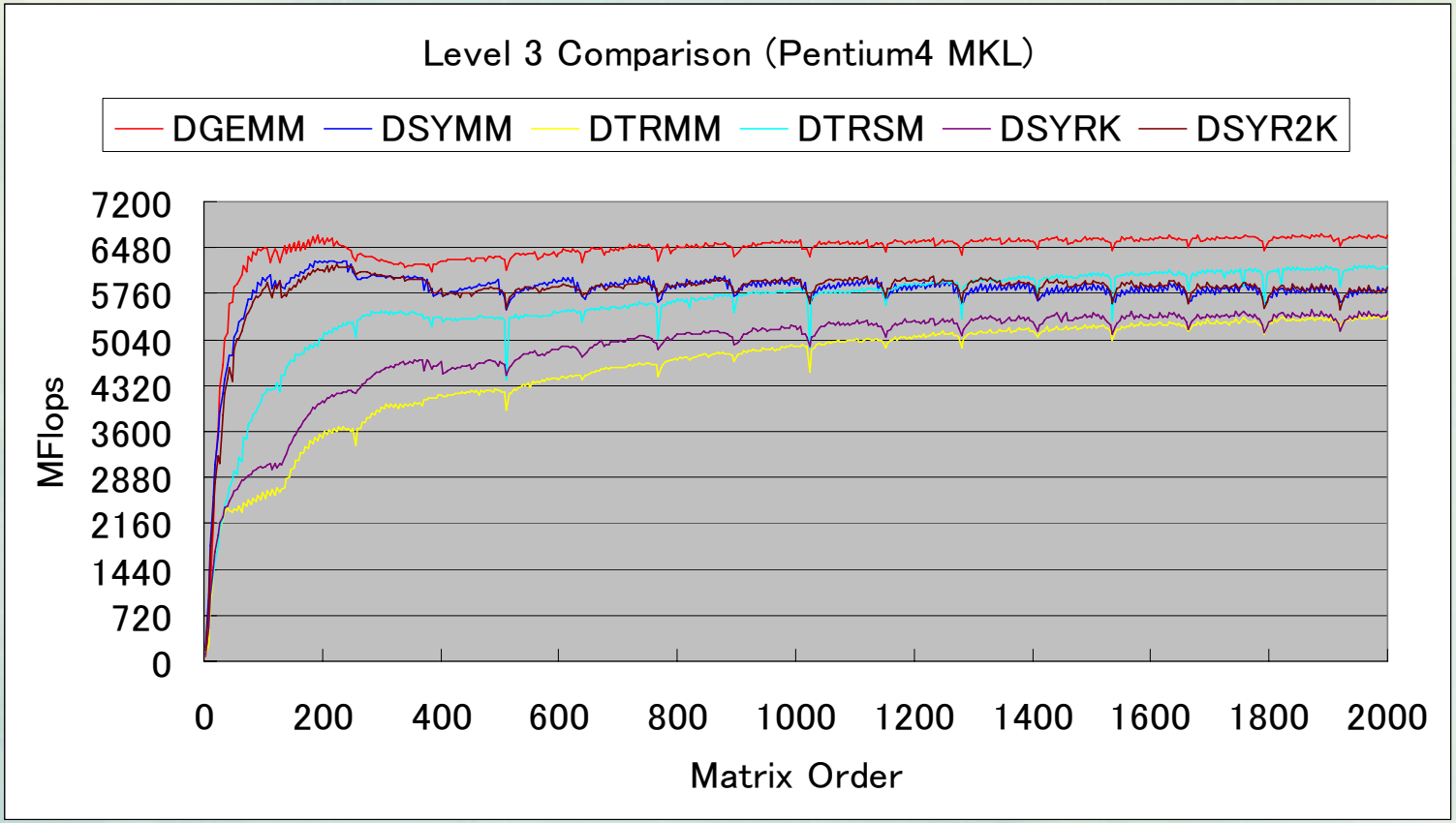
DGEMM (ほとんど一緒)



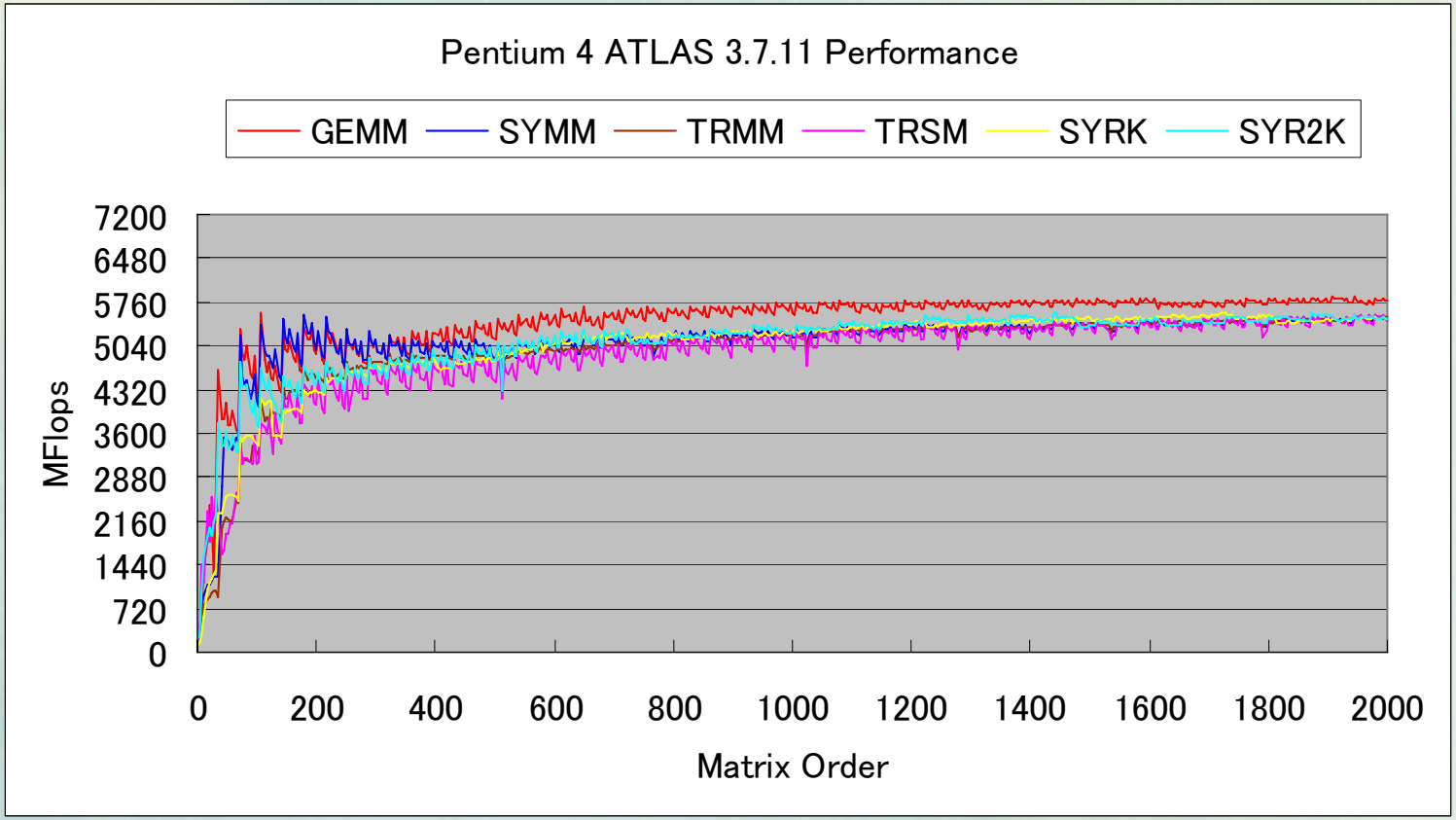
Level 3 Performance (Goto)



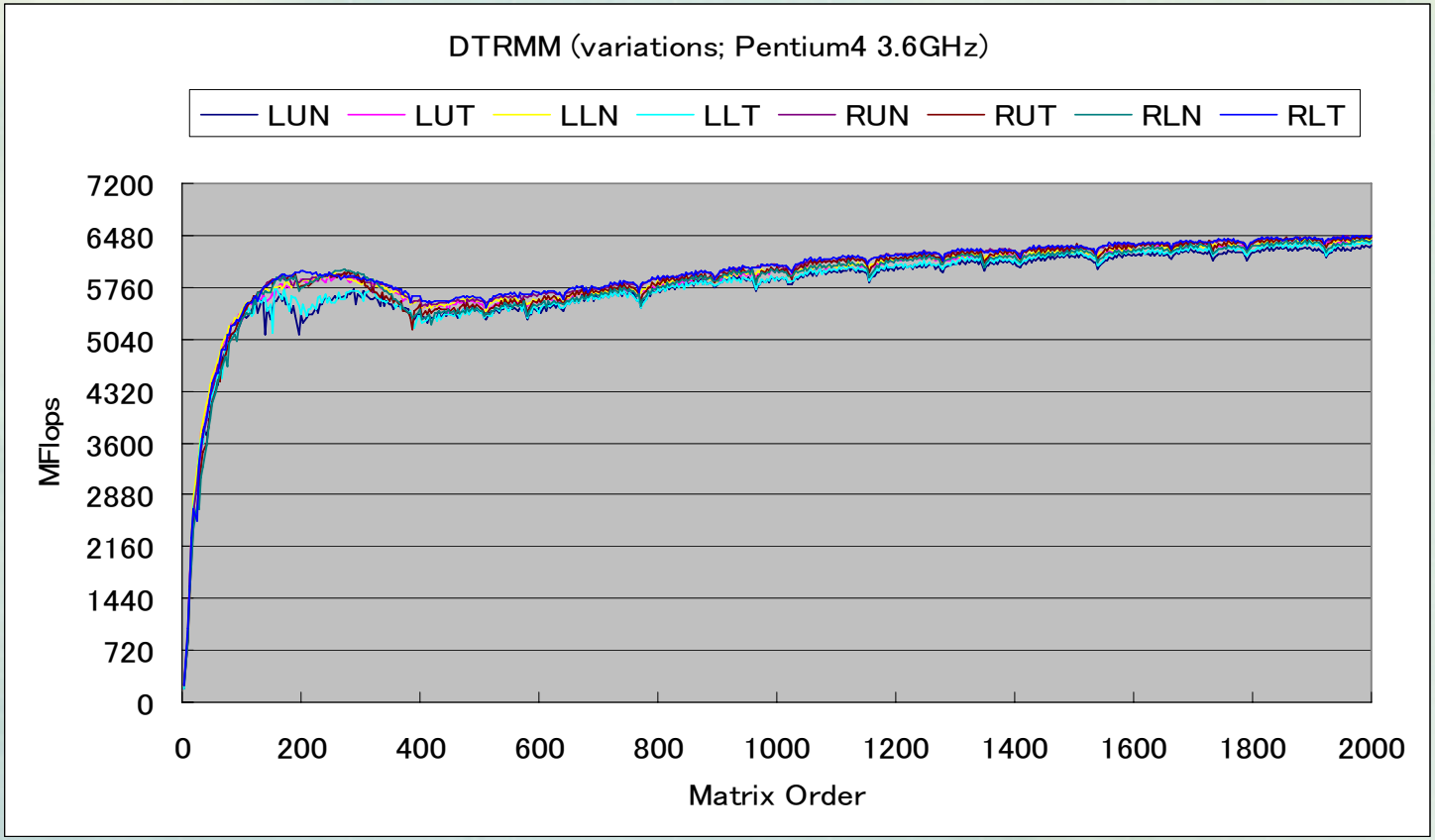
Level 3 Performance (MKL)



Level 3 Performance (ATLAS)



Performance variations (Goto)



まとめ

- 行列積 = コピー + カーネル
- いかにコピーのオーバーヘッドを抑えるかが重要
- コピーのオーバーヘッドを削減して性能を向上させているので、より性能の良いプログラムを作るのはかなり難しい

- Level 3 において小行列を効率よく扱うのは実はかなり難しい

BLAS による最適化の限界

- 常に高速、というわけではない
- 丸投げしていると痛い目にあうかも．．．
  - サイズが小さい場合のオーバーヘッド
  - Incx!=1 時の最適化は真面目にしてない
  - メモリバンド幅の制限、特にストライド
  - バッファの初期化のコスト
  - スレッドの同期オーバーヘッド