複合マルチコア環境のための自動チューニング技術

第2回 自動チューニング技術の
現状と応用に関するシンポジウム
Second symposium on Automatic Tuning Technology and its Application
基盤研究（B） 21300013
「マルチコア複合環境を指向した適応型自動チューニング技術」

今村俊幸

電気通信大学情報理工学研究科

研究目的と体制

マルチコアCPU・Cell・高性能アクセラレータ・GPGPUなど高性能プロセッサアーキテクチャが複雑化することにより、高性能計算を恒常的に維持するための「自動チューニング技術」が現在以上に重要となる。専門家による「匠の技」を科学し一般化することが必要である。これは、次（次）世代スパコンの開発にも大きく寄与する。

本プロジェクトでは適応型自動チューニング＝多様な計算機アーキテクチャの動的な変化や過去の統計情報の取り扱いを研究する。同機能を実現するため

- 各種パラメタを適応かつ自動的にチューニングするためのソフトウェア・フレームワークの確立、
- 同フレームワークを利用したマルチコア複合環境での高性能数値計算ライブラリの実現

の2点に焦点を当てる

【体制】

- 代表

・ 今村 俊幸（電通大） 研究統括・適応型自動チューニング実装方式の研究

- 分担

・ 片桐 孝洋（東京大） 適応型自動チューニング方式の研究

・ 須田 礼仁（東京大） 適応型自動チューニングモデル化の研究

・ 高橋 大介（筑波大） 適応型自動チューニング機能付高速フーリエ変換のチューニング

・ 山本 有作（神戸大） 適応型自動チューニングの概念モデルの研究・同機能付疎行列ソルバの開発

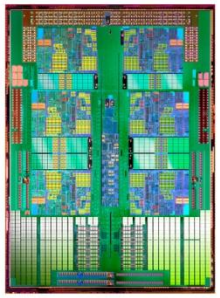
・ 中島 研吾 （東京大） 適応型自動チューニング機能付前処理付反復法ソルバの開発

研究背景

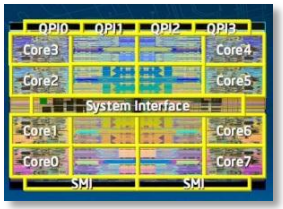
1. マルチコアシステムの普及
2. チューニングの困難さ
3. 市場の速さ

シングルコアチューニング技術はある程度確立している。
マルチコア・メニイコアは途上。シングルコアの技術に対する
プラスアルファ・複合要因はわかっていない！

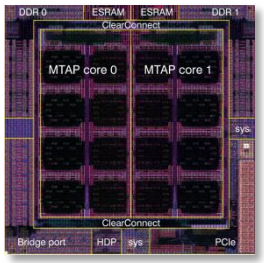
→「適応性」を目指した「自動」チューニング技術の確立



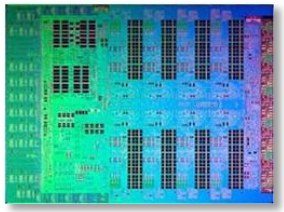
AMD Opteron six-core



Intel Nehalem-EX eight core



ClearSpeed CSX700



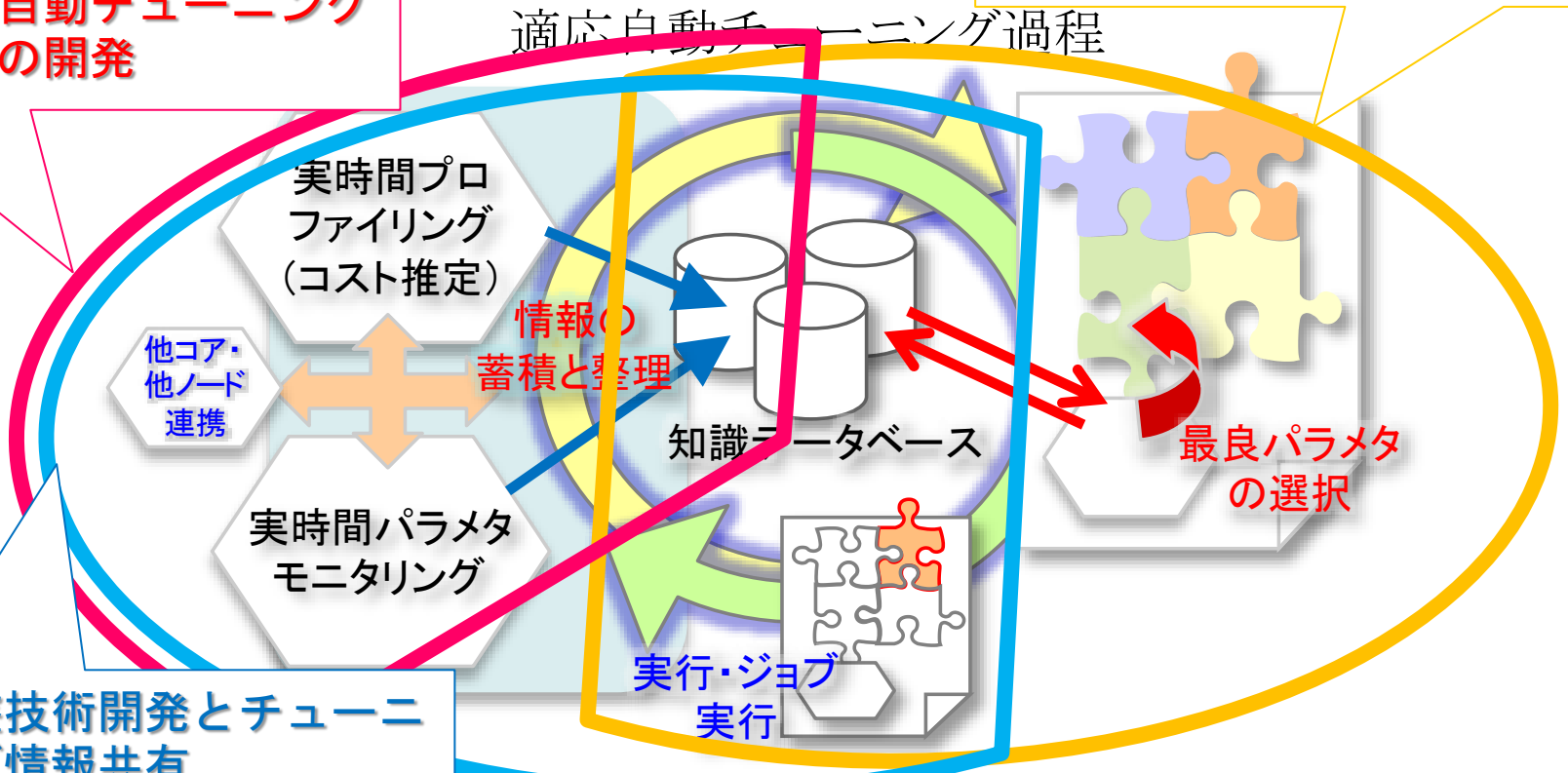
IBM PowerXCell8i processor

適応型自動チューニング方式のフレームワーク

- 適応型自動チューニング方式:
  1. 動的な変化、数の制御、戦略的資源配置
  2. 過去の情報の利用と推測

1. 個々のチューニング技術の開発と蓄積

3. 適応型自動チューニングモデルの開発



2. 基盤技術開発とチューニング情報共有

研究項目 1/4

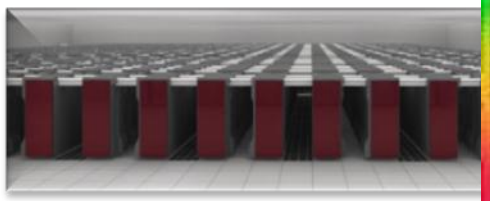
1. マルチコア複合環境でのチューニング技術の開発と蓄積

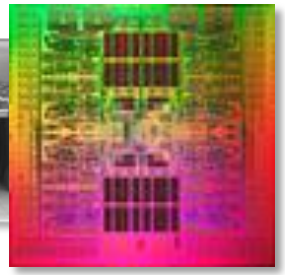
T2K、RICC,「京」、ポストペタ

GPGPU, Cell PS3, PowerXCell8i

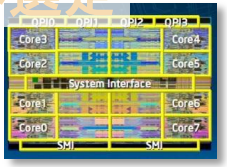
BLAS, FFTW、数値計算ライブラリ などの基本性能測定

→基本チューニング手法やチューニングパラメタ選定

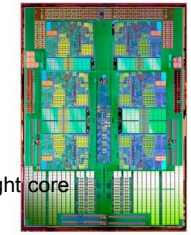




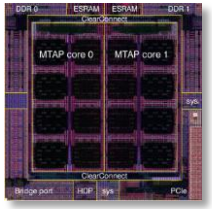
SPARC64 VIIIfx



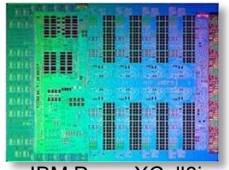
Intel Qnehalem-EX eight core



AMD Opteron six-core



ClearSpeed CSX700



IBM PowerXCell8i processor

T2K Open supercom @ Tokyo

RICC @ Riken

具体的項目 2-4/4

2. 適応型自動チューニング方式の実現、チューニング情報の共有基盤構築
   - チューニングノウハウ、パラメタ
   - チューニング技法の情報化→DB→再利用
3. 適応型チューニングのモデル化
   - 概念モデル構築

実問題への応用
- 疎行列ソルバ
- 前処理付ソルバ
- 固有値計算
- FFT
- MPIの集団通信（Allreduce, alltoall）

1. マルチコア複合資源のチューニング技術の開発と蓄積

GPGPUを用いた固有値計算の可能性

日本応用数理学会2010年年会

GPGPUの倍精度機能が現実に

- Fermi Coreの出現
  - 本格的な倍精度浮動小数点のサポート
  - 理論性能500GFLOPS
- MAGMAプロジェクト
  - Cholesky分解
  - Hessenberg変換
  - その他(LU,QRなど)
  - MAGMABLAS単体も高性能(BLAS準拠APIではない)
    - 300GFLOPS　MAGMABLAS0.3 DGEMM　on Tesla C2050
- CUBLAS (CUDA3.x)
  - 全倍精度BLAS関数をサポート(2.xは倍精度が一部のみ)

Linear Algebra/GPGPU

- 方法．（BLAS->CUBLAS呼び出しのみに限定）
  - Thunking: BLAS->CUBLASがseamlessに
    - 問題点：ホストメモリとデバイスメモリ間の不必要なデータ転送
    - Level 3BLASですら性能は1/2～1/3程度(Level 1は絶望的)

CALL DGEMM('N,','N',M,N,K,alpha,A,LDA,B,LDB,beta,C,LDC)

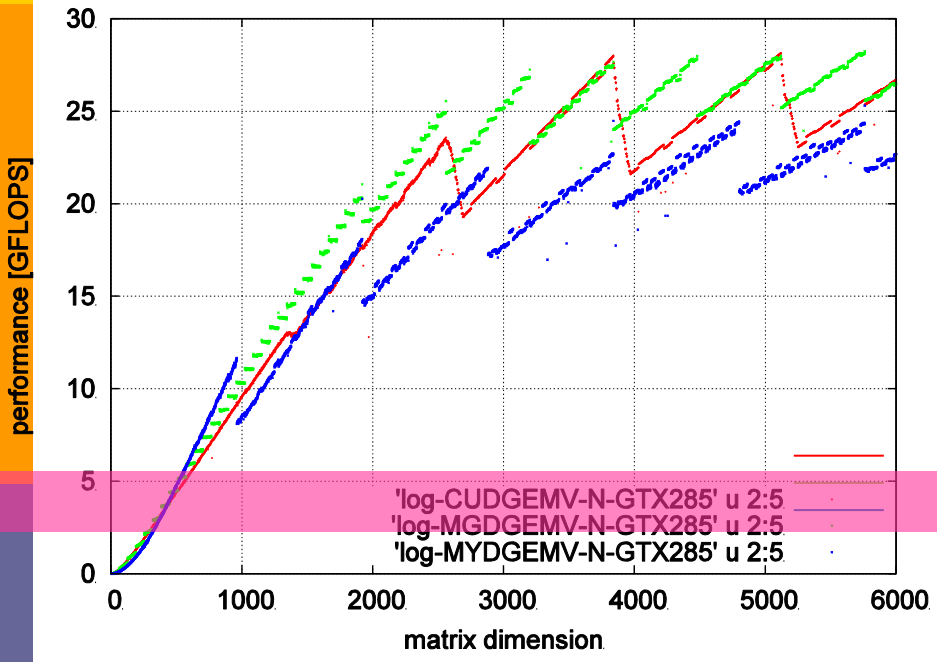
CALL **CUBLAS_DGEMM**('N,','N',M,N,K,alpha,A,LDA,B,LDB,beta,C,LDC)

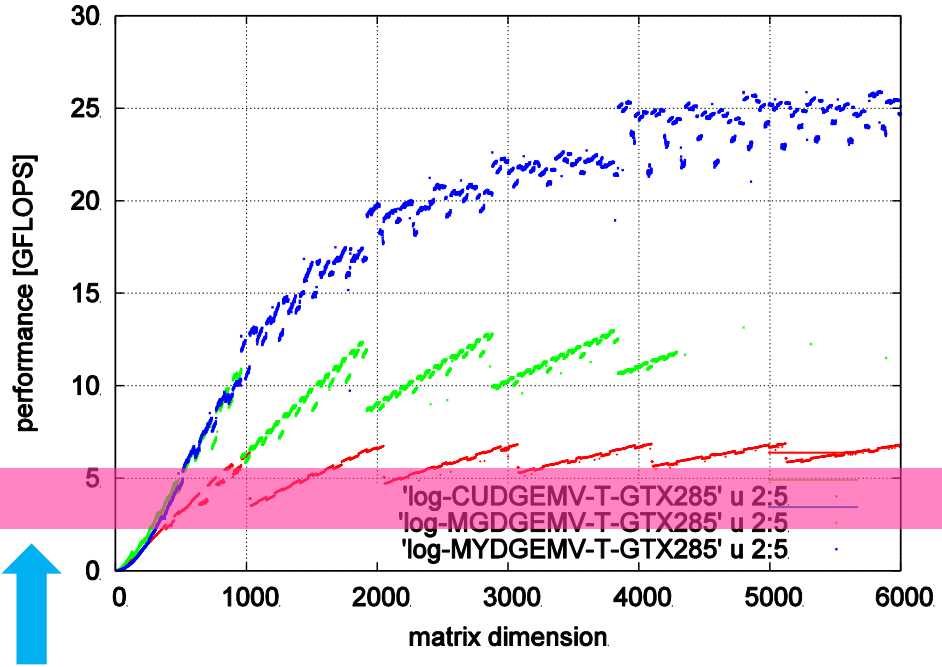
  - Non-thunking
    - ホストーデバイスメモリ間の転送は利用者が制御
    - 不必要なデータ転送はなく、ある程度の性能

基本性能

- DGEMV:

性能はメモリ帯域に依存：標準的なprocessorよりも高帯域→高性能





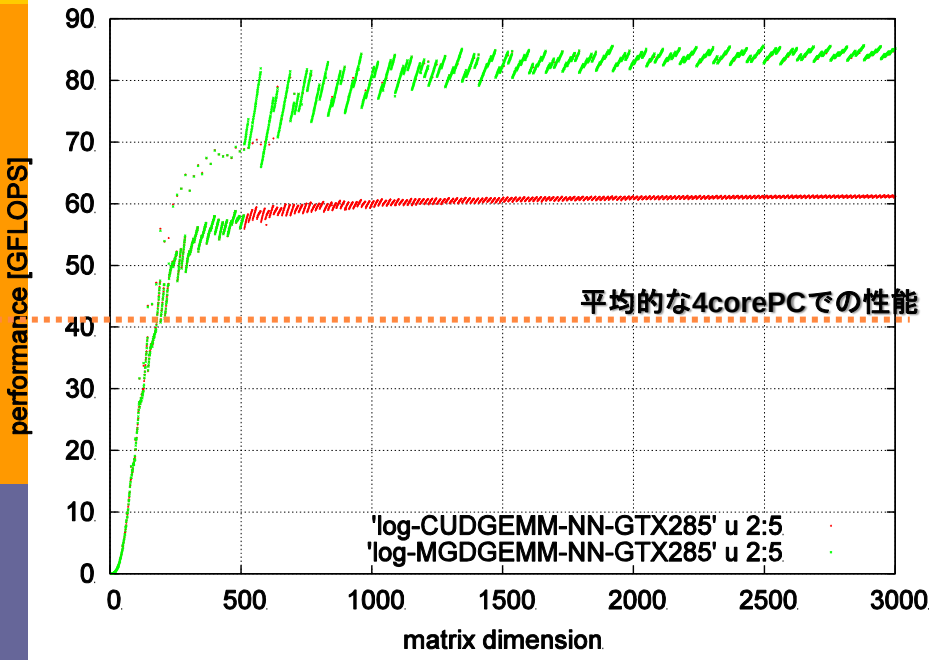
平均的な4corePCでの性能　2~5GFLOPS

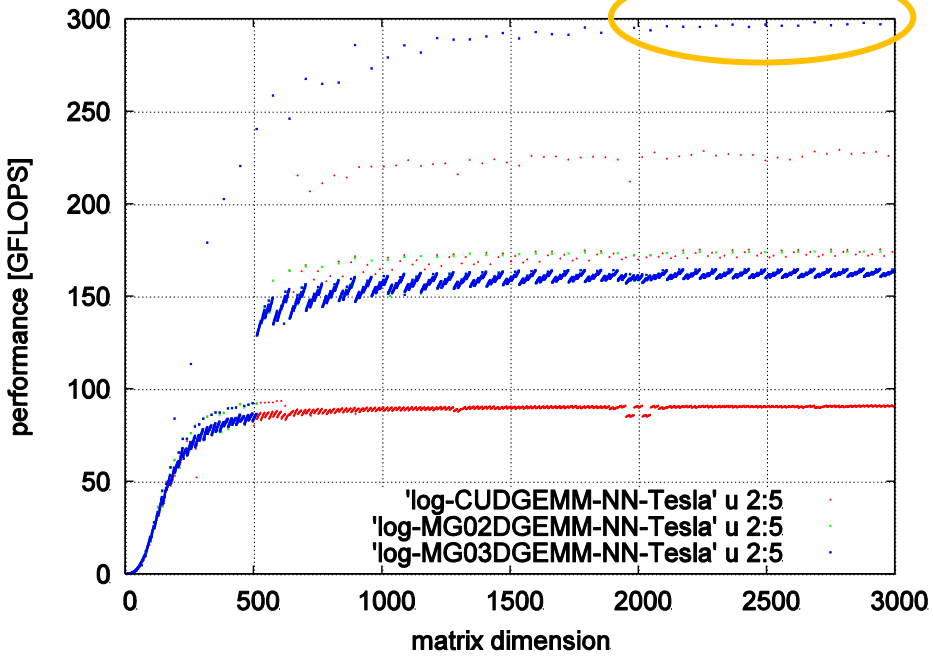
整合寸法は最も近い32の倍数を使用

基本性能

- DGEMM:: 非常によい性能 !!

およそ300GFLOPS!!





整合寸法は最も近い32の倍数を使用

Eigen_sのCUBLAS化

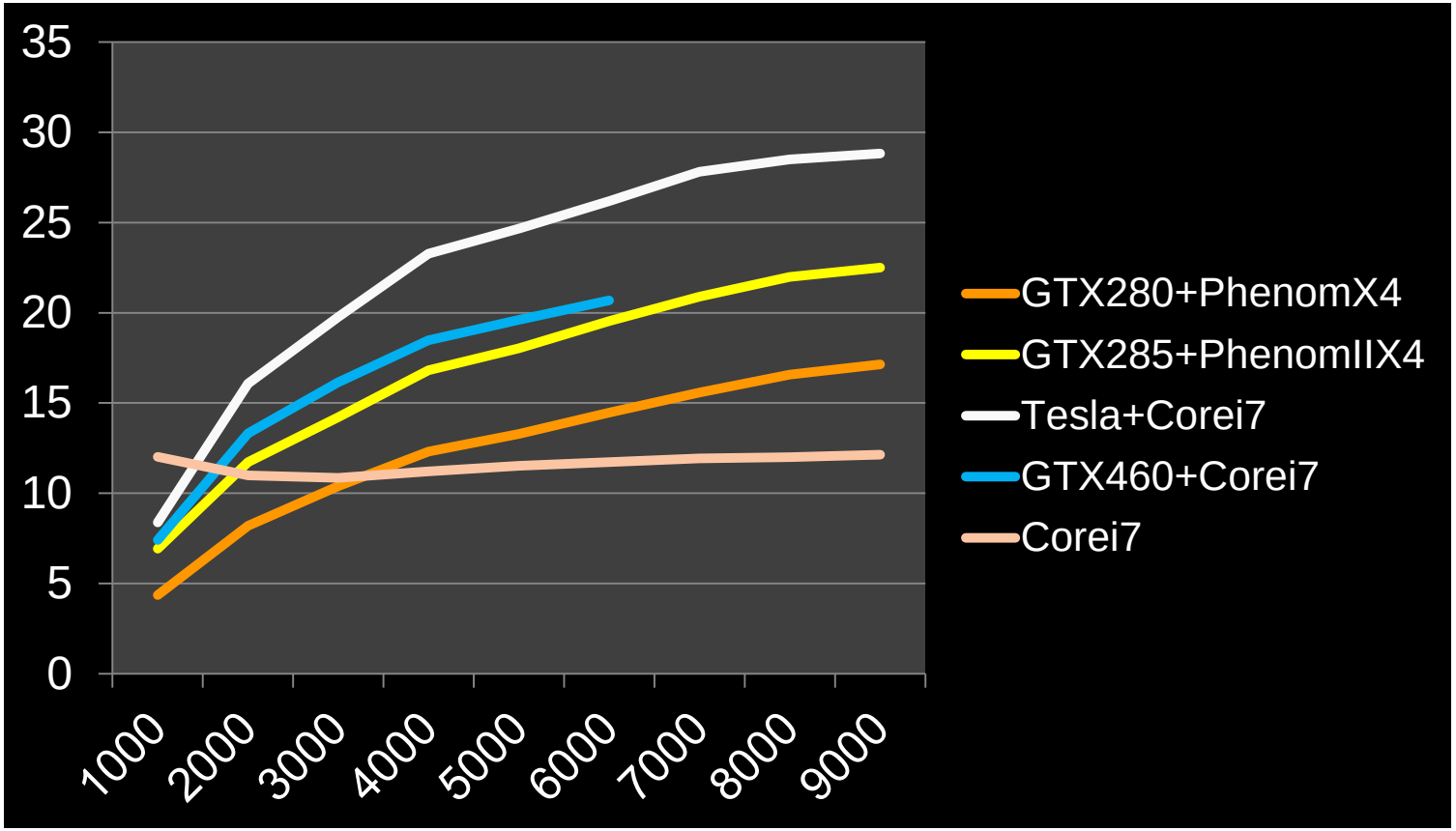
- もともとはマルチコア向け開発版
  - MPI+OpenMP+BLAS
  - BLAS部分からGPUを呼び出す
    - DGEMM, DSYMV相当の関数に適用
    - ベクトル操作はCPU上、データ転送はマスタースレッドで非同期に

CUDAで利用できるhost->device BLAS API

- CUBLAS3.1
- MAGMABLAS0.2
- MAGMABLAS0.3 ([ds]gemm for Fermi)
- MYCUBLAS

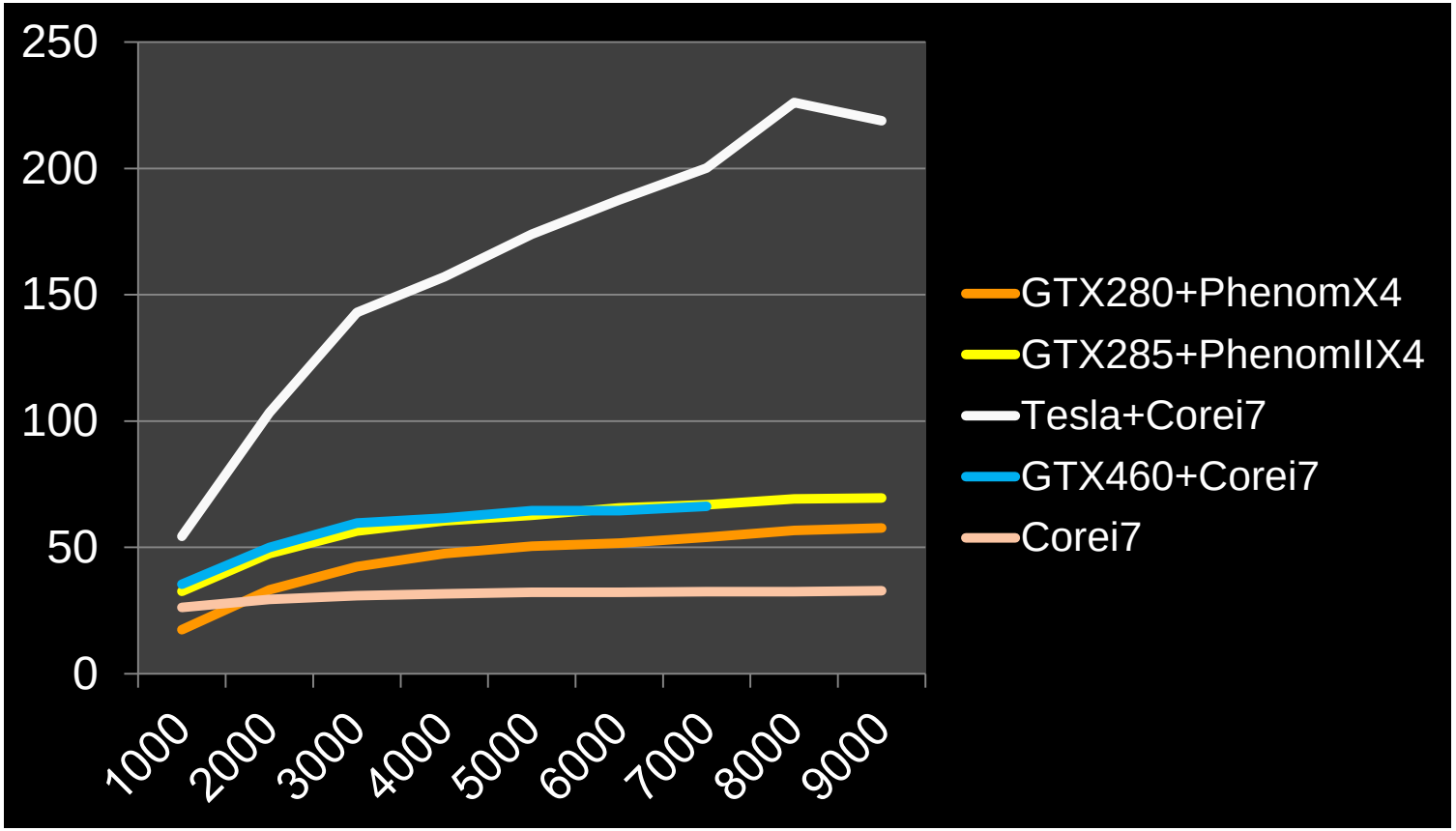
Eigen_s/GPU

- ハウスホルダー順変換(GFLOPS)



Eigen_s/GPU

- ハウスホルダー逆変換(GFLOPS)



Eigen_s/GPU

- Overall computational time [s] on Tesla C2050 +Corei7 860(2.8GHz)

DGEMM: MAGMA
DGEMV: MINE

	CPU only (4cores)	CUBLAS	MAGMA0.2	MYBLAS+ MAGMA0.3	MYBLAS+MAGMA+thunking.dgemm
1000	0.243	0.286	0.275	0.267	**0.298**
2000	1.813	1.397	1.213	1.141	**1.178**
3000	5.857	3.944	3.403	3.032	**2.997**
4000	13.672	8.153	7.088	6.457	**5.841**
5000	25.049	14.783	12.932	11.168	**10.669**
6000	42.326	24.275	21.389	18.076	**17.020**
7000	65.715	37.322	32.472	27.063	**25.429**
8000	101.263	54.234	46.969	38.747	**35.490**

Eigen_s/GPU

- Overall computational time [s] on GTX460+Corei7 920(2.67GHz)

DGEMM: MAGMA
DGEMV: MINE

	CPU only (4cores)	CUBLAS	MAGMA0.2	MYBLAS+ MAGMA0.3	MYBLAS+MAGMA+thunking.dgemm
1000	0.266	0.337	0.325	0.317	**0.346**
2000	1.866	1.733	1.558	1.463	**1.533**
3000	5.972	5.227	4.471	4.025	**4.114**
4000	13.483	11.408	9.517	8.531	**8.452**
5000	25.423	21.486	17.675	15.585	**15.649**
6000	43.120	36.108	29.627	25.604	**25.541**

GPGPU化まとめ

- GPGPUを使って倍精度の固有値計算は1GPUの追加で2から3倍程度の性能向上
  - もっとも単純なCUBLAS方式
  - Level 2,3のみ
- Thunking呼び出しにより不確定なdgemm呼出しに対応したが
  - 小規模問題ではかえって遅くなる、性能不安定性
  - コストモデルによりCPU/GPU/CPU+GPU利用の切り替え
  - 先行関連研究：大島ら、遠藤らなど
- CUBLAS以外もCUDAプログラミングで対応可能だが
  - CPU/GPU/CPU+GPUいずれがよいかは機種・問題依存
  - 似た環境の最適化情報が利用可能
  - → 適応型チューニング
- 別のシナリオ：マルチGPU環境やクラスタ
  - 常にfull-set使用がよいのかは問題依存
  - 過去の情報に応じて適応型チューニング

for $j = N, \ldots, 1$ **step** $-M$

$U \leftarrow \emptyset, V \leftarrow \emptyset, W \leftarrow A_{(*,j-M+1:j)}$

for $k = 0, \ldots, M-1$ **step** K

(1) Householder block reflector: $(C, U^{(k)}) := H(W_{(*,j-k)})$

(2) Matrix-Vectors multiplication (BLAS2.5)

$V^{(k-\frac{2}{3})} \leftarrow A_{(1:j-k-1,1:j-k-1)}U^{(k)}$

(3) $V^{(k-\frac{1}{3})} \leftarrow V^{(k-\frac{2}{3})} - (UV^T + VU^T)U^{(k)}$

(4) $V^{(k)} \leftarrow V^{(k-\frac{1}{3})}C^T - U^{(k)}S, S = \frac{1}{2}C{U^{(k)}}^T V^{(k-\frac{1}{3})}C^T$

$U \leftarrow [U, U^{(k)}], V \leftarrow [V, V^{(k)}].$

(5) $W_{(*,j-k:j)} \leftarrow W_{(*,j-k:j)} - (U^{(k)}{V^{(k)}}^T + V^{(k)}{U^{(k)}}^T)_{(*,j-k:j)}$

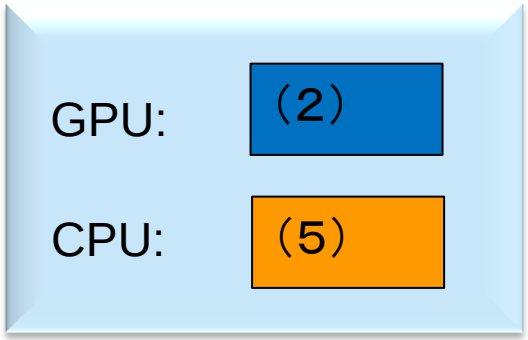
endfor

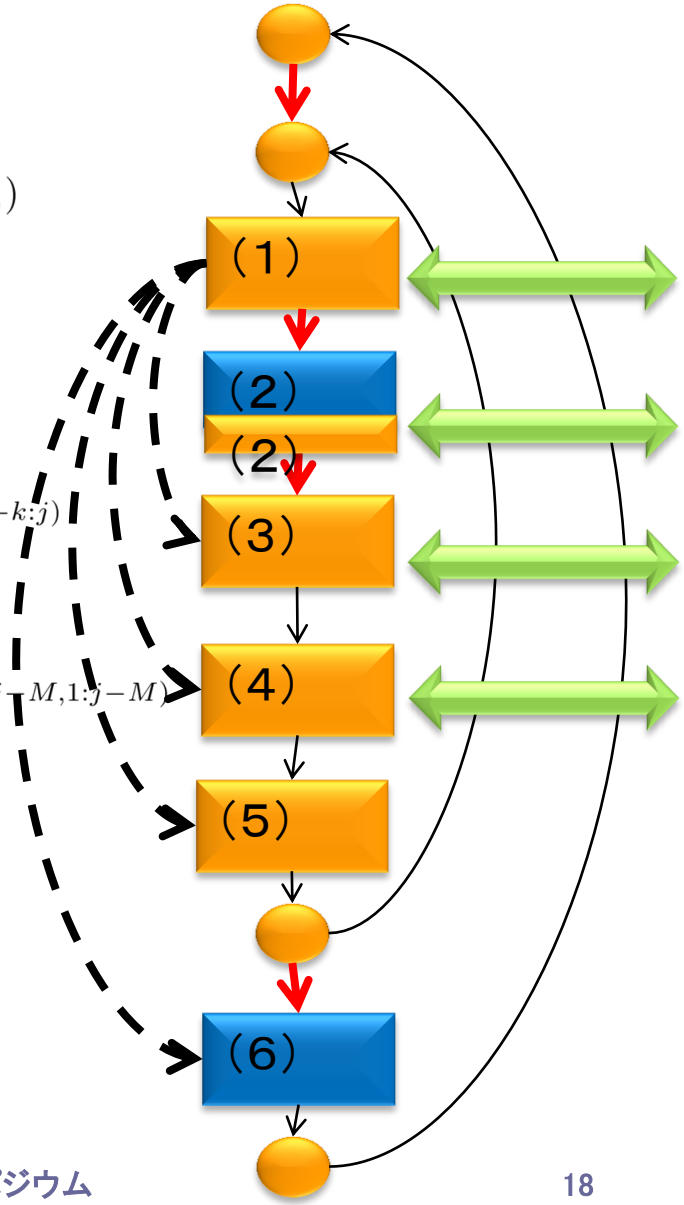
$A_{(*,j-M+1:j)} \leftarrow W$

(6) $2M$ rank-update (BLAS3)

$A_{(1:j-M,1:j-M)} \leftarrow A_{(1:j-M,1:j-M)} - (UV^T + VU^T)_{(1:j-M,1:j-M)}$

endfor





GPGPU化まとめ

- GPGPUを使って倍精度の固有値計算は1GPUの追加で2から3倍程度の性能向上
  - もっとも単純なCUBLAS方式
  - BLAS2,3のみ
- Thunking呼び出しにより不確定なdgemm呼出しに対応したが
  - 小規模問題ではかえって遅くなる、性能不安定性
  - コストモデルによりCPU/GPU/CPU+GPU利用の切り替え
  - 先行関連研究：大島ら、遠藤らなど
- CUBLAS以外もCUDAプログラミングで対応可能だが
  - CPU/GPU/CPU+GPUいずれがよいかは機種・問題依存
  - 似た環境の最適化情報が利用可能
  - → 適応型チューニング
- 別のシナリオ：マルチGPU環境やクラスタ
  - 常にfull-set使用がよいのかは問題依存
  - 過去の情報に応じて適応型チューニング

本年度報告のまとめ

- マルチコア複合環境での適応型自動チューニング技術の確立に向けて
  - チューニング技術の蓄積
  - 複合環境のチューニング問題の例
  - モデル化技術・基盤技術の開発

- 今後の課題
  - チューニング情報のデータベース化
  - チューニング情報の再利用
  - 環境の変化を想定した「適応」型チューニング方式のモデル確立

  など

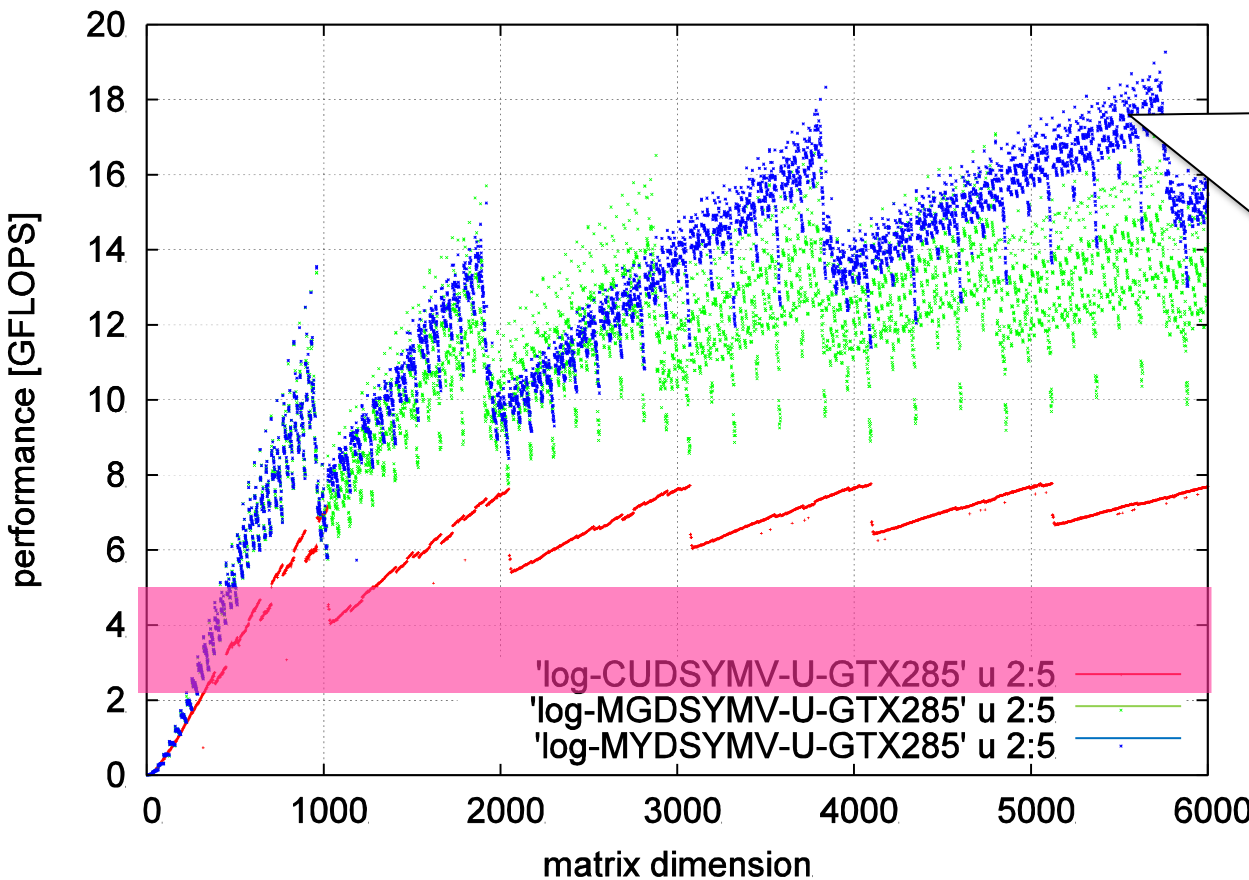
Eigen_s/GPU

- Overall computational time [s] on GTX285+Phenom II X4 945(3GHz)

	CPU only (4cores)	CUBLAS	MAGMA0.2	MYBLAS+ MAGMA0.2	MYBLAS+MAGMA+thunking.dgemm
1000	0.534	0.397	0.351	0.355	**0.369**
2000	2.154	2.068	1.687	1.662	**1.701**
3000	6.870	5.840	4.569	4.530	**4.525**
4000	15.511	12.477	9.609	9.253	**9.066**
5000	29.759	22.888	17.428	16.825	**16.496**
6000	49.642	37.759	28.466	27.031	**26.266**
7000	77.756	58.217	43.639	40.683	**39.510**
8000	115.174	84.649	62.916	57.820	**55.635**

基本性能

- DSYMV:



行列を4分割し、GEMVとSYMVを再帰的に呼出。できるだけDGEMVの高い性能を引き出す。但し、SYMVの性能の影響（大きな揺れ）あり。

整合寸法は最も近い32の倍数を使用

eigen_sx, Narrow-Band-Reduction

for $j = N, \ldots, 1$ **step** $-M$

$U \leftarrow \emptyset, V \leftarrow \emptyset, W \leftarrow A_{(*,j-M+1:j)}$

for $k = 0, \ldots, M-1$ **step** K

(1) Householder block reflector: $(C, U^{(k)}) := H(W_{(*,j-k)})$

(2) Matrix-Vectors multiplication (BLAS2.5)

$V^{(k-\frac{2}{3})} \leftarrow A_{(1:j-k-1,1:j-k-1)} U^{(k)}$

(3) $V^{(k-\frac{1}{3})} \leftarrow V^{(k-\frac{2}{3})} - (UV^T + VU^T)U^{(k)}$

(4) $V^{(k)} \leftarrow V^{(k-\frac{1}{3})} C^T - U^{(k)} S, S = \frac{1}{2} C {U^{(k)}}^T V^{(k-\frac{1}{3})} C^T$

$U \leftarrow [U, U^{(k)}], V \leftarrow [V, V^{(k)}].$

(5) $W_{(*,j-k:j)} \leftarrow W_{(*,j-k:j)} - (U^{(k)} {V^{(k)}}^T + V^{(k)} {U^{(k)}}^T)_{(*,j-k:j)}$

endfor

$A_{(*,j-M+1:j)} \leftarrow W$

(6) $2M$ rank-update (BLAS3)

$A_{(1:j-M,1:j-M)} \leftarrow A_{(1:j-M,1:j-M)} - (UV^T + VU^T)_{(1:j-M,1:j-M)}$

endfor

Narrow-band Reduction algorithm

- Blocking algorithm
- Displace M*v products into M*M.
- It has Trade-off